パソコンプログラマのためのクリエイティブマガジン
平成3年4月2日第三種郵便物認可 1992年3月1日発行(毎月1回18日) 第2巻第3号通巻15号

月刊プログラマーズ・ページ

Programmer's Page

1992 3

定価980円

特集 Cじゃなくてもいいじゃない

C言語を考え，プログラミングへの理解を深めるために

付録:5"2HDフロッピーディスク

本誌掲載プログラム／綴じ込み付録テキストファイル（MS-DOS Version 5.0ファンクションリクエスト一覧）／クイズ・ランダムスクリーン解読ツール 列車発車時刻案内板「TRAINS」

SE
Shoeisha Co., Ltd.

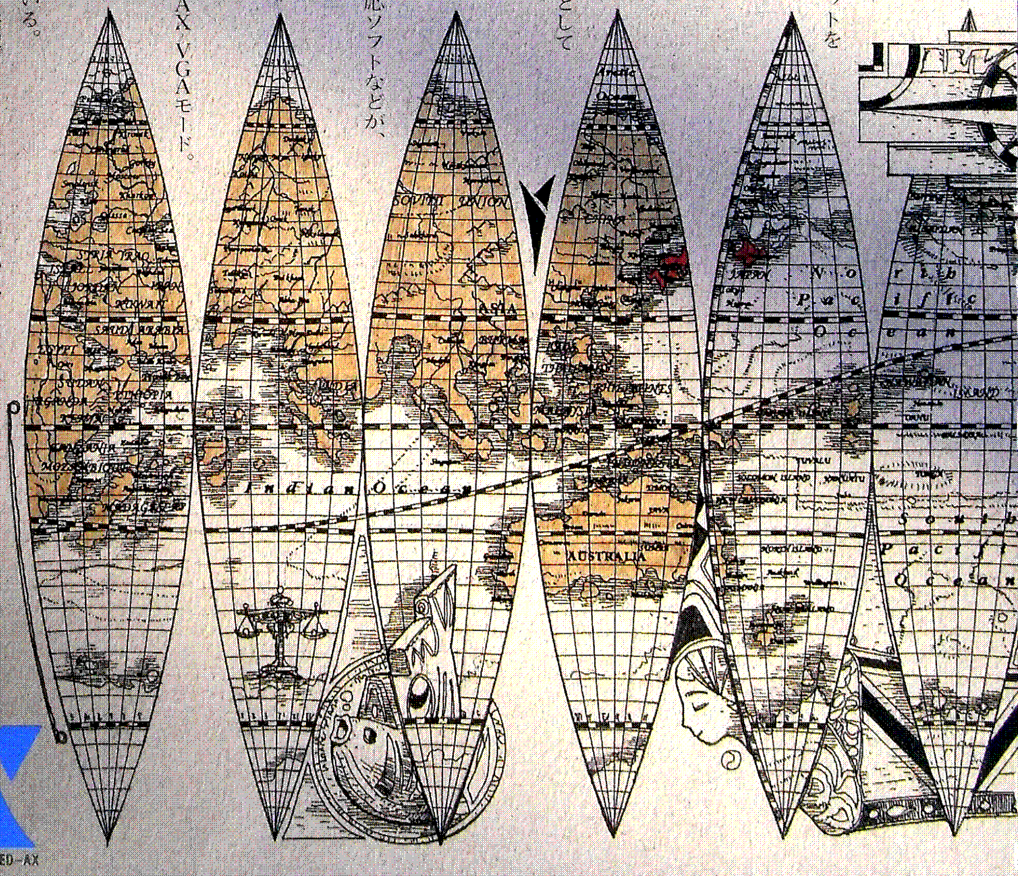
それはハードにとらわれることなく
自由にソフトを選択できるというユーザーの幸運であり、
開発にコストやリスクを背負うことなく、膨大なマーケットを
ターゲットにできるというメーカーの幸運でもあった。
とうの昔に、世界中がそんな幸運一色に染まっている中、
なぜ日本だけが取り残されてしまったのだろうか。

苦難の産物

Windows、LAN、データベース——今日、日本で
使われているソフトのほとんどは、海外でIBM PC用として
開発されたソフトの移植版という、苦難の産物である。
この現実を直視するとき、もはやその不自然さを
無視することはできない。
AXの出現は、そんな不自然さへの解答であった。
海外の最新VGA対応ソフト、MS-Windows3.0対応ソフトなどが、
何のタイムラグもなしに使えることをAXは証明した。
それは不自由であることに慣らされていた者たちにとって、
新鮮な感動だった。
美しい動画像、
アウトラインフォントを実現する640×480ドットのAX VGAモード。
GUI環境を提供するMS-Windows3.0。

マルチメディアの巨大なうねり

今、マルチメディアの巨大なうねりが地球上に広がっている。
そして、ネットワーク・コンピューティングを可能にする共通プラットフォームの実現は、
日本市場が海外市場と同時進行する日が近いことを示す。
AX。——それは、来たるべき時代を乗り越える
日本の世界標準機。



THE ARCHITECTURE EXTENDED-AX

世界の標準機を、日本の標準機に。
我々が、パソコンの新しい時代を拓きます。

Acer Canon NCR OKI olivetti SANYO SHARP

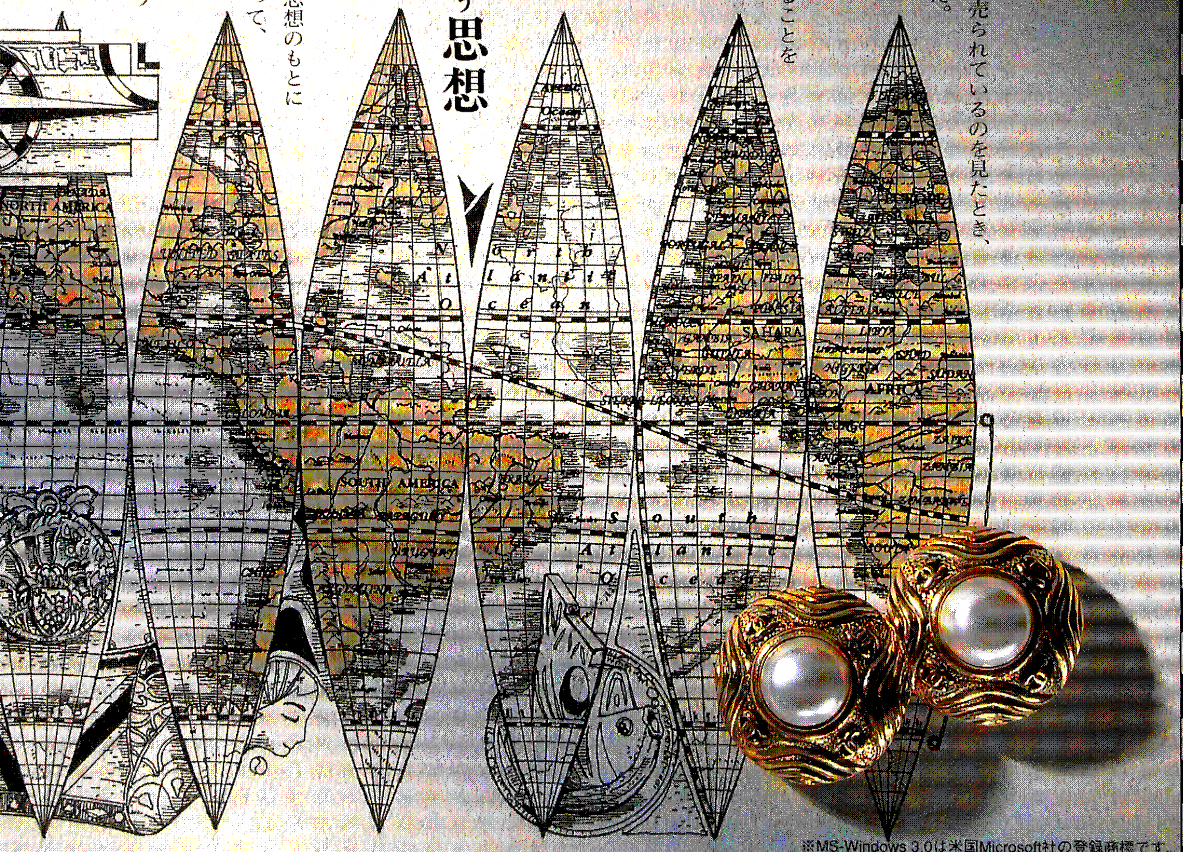
世界標準物語　第二話

シャネルの幸福

パリの路地裏で自分の服のコピーがわずか100フランで売られているのを見たとき、ココ・シャネルは怒るどころか、勝ち誇ったように言った。
「ほらね、私の人生は成功だったのよ。
誰もが、世界でいちばんエレガントで着やすい
私の服を欲しがっているわ」
またある時、シャネルの服のコピーが世界中に溢れていることを
どう思うかという記者の質問に対し、
彼女はこう答えた。
「多くの女性が私の服を着てくれるってことは、
それだけ種々雑多な服が生まれるってことよ」
一生涯、実業家である前に服飾デザイナーである姿勢を
貫き通したシャネルにとって、
それは真実の思いであったに違いない。

オープンアーキテクチャという思想

IBM PCが、その誕生からわずか数年で
事実上の世界標準機になったいきさつを考えるとき、
ふと、そんなシャネルの逸話を思いおこす。
IBM PCは、オープンアーキテクチャという開かれた思想のもとに
生まれたため、世界中のハード／ソフト・メーカーは競って、
互換機や周辺機器、ソフトの開発にのりだした。
その結果、安価で豊富な互換機が市場に溢れ、
現在6万本以上といわれるソフト資産が蓄積されるという
幸運な事態となった。



※MS-Windows 3.0は米国Microsoft社の登録商標です。
※IBM PC、VGAは、米国IBM社の商標です。

CASIO　HITACHI　KYOCERA　MITSUBISHI
SONY　UNISYS　YHP 横河・ヒューレット・パッカード

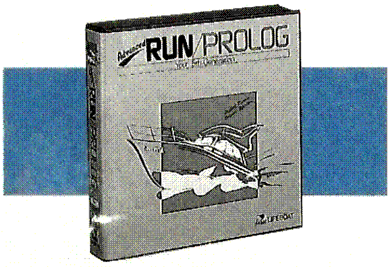
社名アルファベット順

お問合わせ：株式会社 翔泳社
TEL. 03-3263-0447
〈資料請求番号P02〉

Advanced RUN/PROLOG

対応機種：NEC PC-9801シリーズ(98NOTE対応)
東芝 J-3100シリーズ(Dyna Book対応)
富士通 FM-R50/60/70(TOWNS対応)
IBM PS/55シリーズ(S/T/V/Zモデル)
メモリ：512KB以上

ハイスペックPROLOGインタープリタ シリーズ統一価格 ¥29,800



論理型言語と呼ばれ、人間の思考パターンに非常に近い言語仕様を持つPrologは、LISPと並んでAI処理の研究・開発に最も適した言語です。Advanced RUN/PROLOGは、自然言語解析パーサDCGの搭載による文法解析機能、専用の対話型デバッガによるソースラインデバッグ、仮想データベース機能による最大4メガバイトの広大なデータベースなど、入門用としては異例のハイ・スペックを装備しています。

4メガバイトの仮想データベース

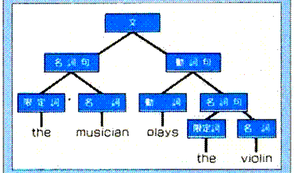
知識処理などの人工知能プログラムには、大量の記憶容量が不可欠です。Advanced RUN/PROLOGは、4メガバイトの仮想データベース空間を持っており、マシンのRAMが512Kバイトでも、ディスク上に4メガバイトまでデータベースを拡張できます。しかも、そのための手続きなどは一切必要ありません。

自然言語解析パーサDCG(限定節文法)

通常のProlog構文を拡張したDCGは、言語解析及び言語記述に優れた能力を持っています。通常、言語（自然言語、プログラム言語など）というものは、ある一定の規則をもって構成されています。これをPrologで記述・解析するためのパーサーがDCGです。これを使うことにより、ユーザーインターフェースの実現から、コンパイラ、プログラムジェネレータの作成まで様々な開発を効率的に実現できます。

```
文 - -> 名詞句(_数詞, 人), 動詞句(_数詞).
名詞句(_数詞, _対象) --> 限定詞, 名詞(_数詞, _対象).
動詞句(_数詞) --> 動詞(_数詞), 名詞句(_数詞, 物).
限定詞 - -> (the).
名詞 (単数, 人) --> (musician).
名詞 (複数, 人) --> (musicians).
名詞 (単数, 物) --> (violin).
動詞 (単数) --> (plays).
動詞 (複数) --> (play).
```

●DCG記法



●生成されるパースツリー

エジンバラ文法完全準拠

従来より数多くのアプリケーションプログラムの開発・実行がおこなわれてきたDEC-10 Prologに準拠していますので、多くの入門書、解説書がそのまま利用できます。

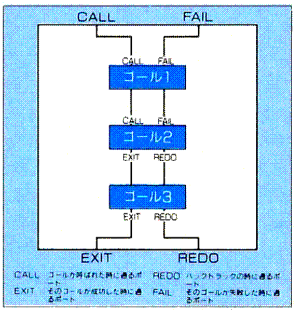
Arity Prologへの移行が可能

Advanced RUN/PROLOGは、本格的な汎用開発向け処理系であるArity/Prologのインタープリタ部分をもとに開発されています。ちがいはインタープリタであることと、拡張機能であるB-treeが省略されていること、仮想記憶がArity/Prologでは1ギガバイトあることです。したがって、Prologとしてはフルスペックの機能を持っています。もちろんArity/Prologとプログラムには互換性があります。学習あるいは本格導入の検討にAdvanced RUN/PROLOGを用い、システム開発段階で、Arity/Prologコンパイラに移行することができます。

ソースレベルデバッガ

Prologのプログラムは、手続き型言語と違ってバックトラックしていくために、従来のようにデバッグするわけにはいきません。そのためにArity Prologでは、"Clocksin とMellishがProgramming in Prolog"で述べているプロシジャーボックスモデルと呼ばれるデバッガを用意しています。

●プロシジャー・ボックスモデルデバッガ



CALL　コールが呼ばれた時に通るポート
EXIT　そのコールが成功した時に通るポート
REDO　バックトラックの時に通るポート
FAIL　そのコールが失敗した時に通るポート

Advanced RUN/LISP

対応機種：NEC PC-9801シリーズ(98NOTE対応)
東芝 J-3100シリーズ(Dyna Book対応)
富士通 FM-R50/60/70(TOWNS対応)
IBM PS/55シリーズ(S/T/V/Zモデル)
メモリ：640KB以上

ハイパフォーマンスCommon LISPインタープリタ

AI言語LISPの集大成であるCommon LISPに準拠したインタープリタであるAdvanced RUN/LISPは、最大8メガバイトの仮想記憶をディスク上に確保することで、メモリ容量の限られたMS-DOS上でのAI研究・開発を可能にしました。トレース機能、ステップ実行機能、エラー・ハンドラ等、デバッグ機能を特に強化し、AI言語LISP入門を初めて一般化したLISP入門学習環境です。

LISP環境の実現

LISPはもともと高度に対話的で優れたプログラミング環境をもっています。Advanced RUN/LISPではさらに、トレーサ、エラー・ハンドラ、ステップ実行など、プログラミング／学習効率を大幅にアップする拡張機能が追加されています。

Common LISPデバッグ用関数：ERROR、DESCRIBE、TRACE、UNTRACE、STEP

Common LISP準拠

Common LISPに準拠した約400個の関数、スペシャルフォーム、マクロをサポートしています。またキーワードの大部分とシステムが用意する4つのパッケージもサポートしているので、上位機種で蓄積されたCommon LISPアプリケーションの移植も容易です。

8メガバイトの仮想記憶

LISP言語が長い歴史を持ちながら、パソコン上で満足な処理系の実現が難しかったのは、実行時に大きなメモリ空間を必要とすることが原因でした。Advanced RUN/LISPでは、最大8メガバイトの仮想記憶をディスク上に確保することによって、パソコン上でのLISPの最大のネック、メモリ・ハングリーを解決しました。

独自機能を拡張

Advanced RUN/LISPでは、Common LISPの中で規定されている関数の他にMS-DOS上での仕様を考慮した独自の機能を追加しています。追加された機能の中には、MS-DOSのコマンドを実行する為のマクロや、独自のエディタを設定するコマンドや、エディタによって編集されたプログラムを即座に読み込む為のコマンドなどが含まれています。

●step実行

```
LISP: (step '(factorial 20))
|| 2 <enter> FACTORIAL 20
STEP 0> ...
 ...
>>>STEP 3> ?
------------       STEP USAGE
<Return>     - ワンステップ実行
b            - 実行経路の簡略表示
B            - 実行経路の詳細表示
f または F   - 各ステップで停止しない
#            - フレーム数表示
n[num]       - numステップ以前の関数表示(default 1)
N[num]       - numステップ以前のフレーム表示(default 1)
p[num]       - numステップ以降の関数表示(default 1)
P[num]       - numステップ以降のフレーム表示(default 1)
s            - 先頭の関数表示
S または >   - 先頭のフレーム表示
t            - 最新の関数表示
T または <   - 最新のフレーム表示
q, Q, x, X   - ステップ実行中止
?            - ヘルプ画面表示
------------
STEP 3> b
```

【お求め方法】当社製品は、パソコンソフト取扱い店でお求めいただけますが、弊社に直接ご注文の際は、The Programmer's INDEX(TEL:03-3293-3887)までお問い合わせ下さい。
※各製品の表示価格には、消費税は含まれておりません。

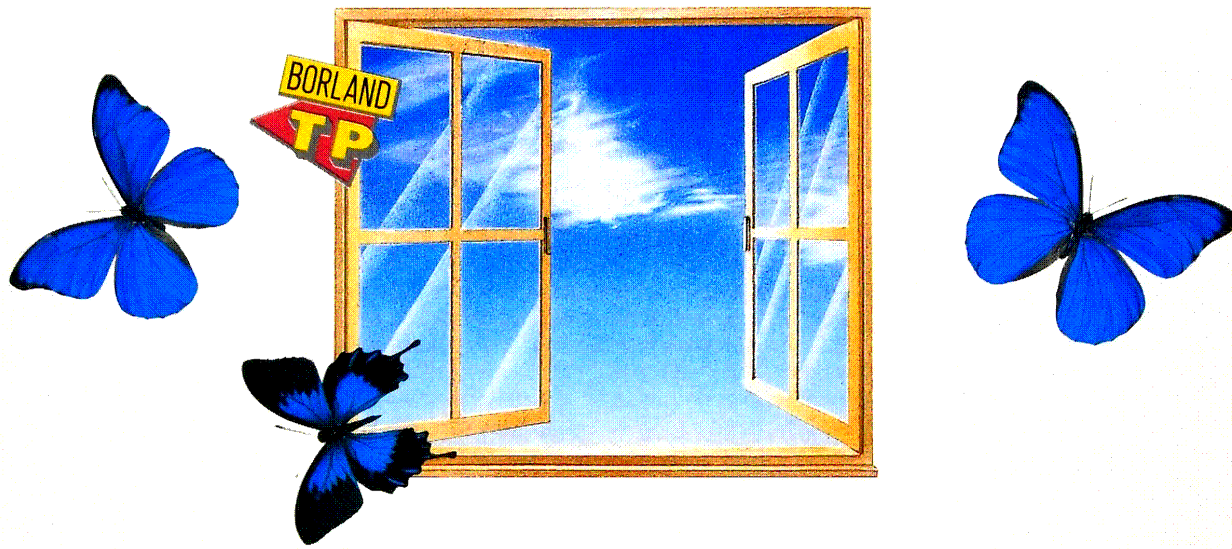
●Copyright Notice● ＊Advanced RUN/LISPは、株式会社LIFEBOATの商標です。＊Arity Prologは米国Arity Corpの登録商標です。＊Advanced RUN/PROLOGは株式会社LIFEBOATと米国Arity Corpとの共同開発製品です。

株式会社 LIFEBOAT
〒101 東京都千代田区神田錦町3-6
PHONE:03-3293-4711(代) FAX:03-3293-4710

〈資料請求番号001〉

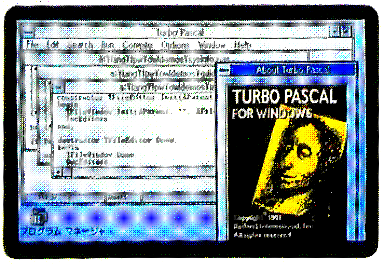
BORLAND
The Leader in Object-Oriented Programming

"Easy & True" for Windows.

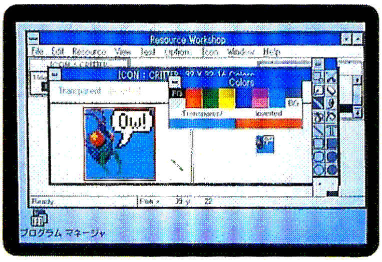


基本は、ターボパスカル――。
完ぺきなWindowsアプリケーション開発環境を提供。

その美しい言語仕様から"コンパイラの貴婦人"と呼ばれ、多くのプログラマーを魅了してきたパスカルが今、「Turbo Pascal for Windows」として生まれ変わりました。デビュー当時、その驚異的な低価格とコンパイルスピードの速さ、デバッガやエディタを備えた世界初の統合環境で人々を圧倒し、瞬く間に標準の地位を確立して以来、ターボパスカルは常に先進の機能を搭載し、言語の世界をリードし続けています。アメリカでは、大学などの教育機関や研究施設でターボパスカルを使用することが常識ですし、ヨーロッパや日本でもビジネスの分野で幅広く利用されています。プログラミングの学習もターボパスカルから始めれば、ベーシックで貧弱な言語で学習したために時間を無駄にする、ということはありません。しかもそのターボパスカルが、複雑で難解だったWindowsのプログラミングを簡単にしてくれるのです。



Turbo Pascal for Windowsには、Object Windowsというアプリケーションの骨格になるクラスライブラリが付属しています。ここからウィンドウやメニュー、ダイアログ、コントロールなどを継承すれば、最初からコードを記述する必要はないのです。オブジェクト指向の拡張がされていないC言語でのプログラムより、格段に素早く、短いコードでアプリケーションを作成できるようになります。Turbo Pascal for Windowsは、自分自身がWindowsのアプリケーションであり、Windowsプログラムのみの作成用に設計されています。



ソースコードの編集、コンパイル、実行、デバッグなど、使いやすさで定評のある統合環境を用い、全てWindows上で行えます。ダイアログ、メニュー、ビットマップ、カーソル、アイコン、文字列リソース、アクセレータ、フォントなどのWindowsリソースを視覚的に作成したり、編集することができるリソースワークショップも付属していますから、高額なSDKを別途購入する必要もありません。Turbo Pascal for Windowsには、Windowsアプリケーション作成に必要なツールの全てが、ひとつのパッケージに含まれているのです。

■動作環境：PC-9800シリーズ（ハイレゾモード対応、LTは除く）、PC-286/386シリーズ/MS-DOS：Ver 3.1以上/MS-Windows 3.0以上必須/マウス：バスマウス/要ハードディスク
■価格……………………29,800円（税別）

NEW

TURBO PASCAL®
FOR WINDOWS

ユーザ向け情報誌
LANGUAGE EXPRESSプレゼント!!
■この広告をご覧になって、製品に関する資料を請求していただいた方全員に、ボーランド発行の情報誌「LANGUAGE EXPRESS」をさし上げます。

※All Borland Products are trademarks or registered trademarks of Borland International, Inc. © Borland International, Inc.
※その他、商品名は一般に各社の商標です。
●製品に関するお問い合せは、ボーランド株式会社 TEL.03-5350-9370までお願いします。

ボーランド株式会社
〒151 東京都渋谷区笹塚1-64-8 笹塚サウスビル TEL.03-5350-9370㈹
※移転いたしました。

〈資料請求番号002〉

次代のアドバンテージ、最新の技術をあなたに！

Microsoft® University

独学より大学。

マルチメディアのプラットフォーム MS-Windows、高信頼性の MS OS/2。本格的なマルチタスクの実現、グラフィカルなユーザ・インタフェース（GUI）、LANマネージャによる高度なネットワーク処理など、技術者の創造力を刺激する環境を提供します。マイクロソフト・ユニバーシティは、その登竜門。ウインドウ環境、デバイスドライバの開発、ネットワーク処理など、アプリケーション・ソフト開発に欠かせない、ノウハウをレクチャします。入門から専門まで、充実したコースラインアップ。短時間で即開発に結びつく新しいコンピューティングスタイルを身につけて下さい。

Microsoft Windows Programming Environment
ウインドウズ・プログラミング・エンバイラメント

■期間：5日間 ■受講料：18万円（税別）

■MS Windows Ver3.0のアプリケーションを作成する方のためのコースです。Windows プログラミングのポイントを重点的に解説、実習中心のコースです。

Microsoft Windows Applications Programming
ウインドウズ・アプリケーションズ・プログラミング

■期間：5日間 ■受講料：18万円（税別）

■MS Windows Ver3.0対応のアプリケーションを開発するプログラマや設計者のためのコースです。ユーザインタフェースやアプリケーションなど設計ポイントがマスタできます。

Microsoft LAN Manager Programming Environment
LANマネージャ・プログラミング・エンバイラメント

■期間：4日間 ■受講料：15万円（税別）

■マイクロソフト LAN マネージャ（LM）用アプリケーションを開発する設計者のためのコースです。LMのアーキテクチャと特徴について解説します。

Microsoft LAN Manager Applications Programming
LANマネージャ・アプリケーションズ・プログラミング

■期間：5日間 ■受講料：18万円（税別）

■マイクロソフトLAN マネージャ（LM）上で動作するアプリケーションの作成や移植するプログラマのためのコースです。

MS OS/2 Presentation Manager Programming Environment
プレゼンテーション・マネージャ・プログラミング・エンバイラメント

■期間：5日間 ■受講料：18万円（税別）

■MS OS/2プレゼンテーション・マネージャのアプリケーションを作成するための入門コースです。作成に必要となる基本事項を説明いたします。

MS OS/2 Presentation Manager Applications Programming
プレゼンテーション・マネージャ・アプリケーションズ・プログラミング

■期間：5日間 ■受講料：18万円（税別）

■MS OS/2 PMのアプリケーションを作成するためのステップアップコースです。経験者の方を対象に、より詳しく説明します。

MS OS/2 Programming Environment
プログラミング・エンバイラメント

■期間：4日間 ■受講料：15万円（税別）

■MS OS/2のアプリケーションを作成するために必要となる基本的技術を修得するコースです。本コースは、OS/2に取り組む上での基本コースとなります。

MS OS/2 Applications Programming
アプリケーションズ・プログラミング

■期間：5日間 ■受講料：18万円（税別）

■MS OS/2のAPIを使用し、マルチタスク、メモリ管理、プロセス間通信、ダイナミックリンクなどの機能を有効に活用する、実習を中心としたコースです。

MS OS/2 Device Drivers
デバイス・ドライバ

■期間：4日間 ■受講料：15万円（税別）

■MS OS/2のデバイス・ドライバの機能、構造を修得し、実際にデバイス・ドライバを作成するコースです。デバイスヘルパールーチンなども解説します。

C言語入門

■期間：2日間 ■受講料：6万円（税別）

■これからC言語を始めようとする方を対象に、C言語の基本的な文法からわかりやすく解説します。統合環境を持つQuick Cを使って理解を深めます。

C言語応用

■期間：3日間 ■受講料：9万円（税別）

■関数、ポインタ、構造体についてプログラムを作成しながら詳しく説明します。セミナは、最新のMicrosoft C Ver 6.0を使用して実施いたします

●お問い合せは

富士ソフトウエア株式会社 MU教育センタ

TEL.03-5461-8296 FAX.03-5461-8469

日本電気株式会社殿の御協力によりPC-9800シリーズ対応コースも好評開催中!!

Microsoft®
マイクロソフト株式会社
〒160 東京都新宿区西新宿7-5-25 Kビルディング

FSI 富士ソフトウエア株式会社
MU教育センタ
〒108 東京都港区港南2-12-27イケダヤ品川ビル

〈資料請求番号003〉

Brief

Full Screen Editor 日本語版

全米のプログラマが認めた実力！
アメリカでは「ブリーフ」が
プログラミング用エディターとして
最も多く利用されています。

C言語で組めるマクロ！

Version UP!

v.3.0 いよいよ登場。
プログラミングエディター新世紀。

よりパワフルで、よりエレガントな プログラミングのために……。

●C言語で組めるマクロ 新機能

使い慣れているC言語でマクロが組めますから、習得が簡単でカスタマイズも思いのままです。

●マクロのスクリーン・デバッガ 新機能

ソースコードでデバックできます。もちろんブレークポイントやウォッチ・ドッグもセットできます。

●テンプレート機能〈構文の自動生成〉 新機能

C、Ada、Basic、COBOL、FORTRAN、MODULA2、Pascalで構文を自動生成します。

●300回ものアンドゥ

各ファイルごとに300までの操作を元にもどすことができますので、やり直しの面倒を気にせず、安心して編集できます。

●安全編集

Briefは、一定時間キーを操作しないこと最新の編集内容をファイルに自動保存します。だから編集中に電源が切れても安心です。

●その他の主な新機能

キー操作のファイルへのセーブとリプレイ、処理可能行数を拡張、リジューム機能、Ctrlキー・アサイン（オプション）、メニュー（オプション）etc.

アメリカでも日本でも高い評価

●プログラミング向きエディターを探している人は、もう探す必要はない。(Jerry Pournelle BYTE)
●ソフト開発には強力なツール、300回のアンドゥは大きな特徴。（日経バイト）
●既存のエディターに不満を覚えたら、これしかない。(Oh! FM)
●まるで夢のようなアンドゥだ――おもわず間違うのを期待してしまいそうだ。(The C Users Journal)

■主な機能

◆55ウィンドウ◆無限のファイル◆自動コンパイル◆自動エラー・ジャンプ◆自動インデント◆自動テンプレート◆バッファ書出 切換 削除 リスト表示◆ファイル名の指定 表示 変更◆バックアップ保存◆特殊文字入力◆2行を1行に連結◆ファイル単位の挿入◆増分検索◆各種マーク 始・終点の入換◆ブロックのカット コピー ペースト 印刷 保存 左右へのシフト◆対応する括弧へ移動◆C言語のコメント化 非コメント化◆DOSコマンドの実行◆キー操作記憶 実行◆入力ヒストリー◆英文テキスト整形◆タブのスペース挿入◆拡張子毎に異なった設定ができるタブ◆検索置換パターンの画面からの入力◆英字の大小文字変換 その他

定価48,000円
（消費税は含ます）

■5機種対応パッケージ

（東芝、NEC、AX協議会、IBM、富士通）

●5"2DD、3.5"2DDデュアルメディア●MS-DOS 3.1以降に対応●英語DOSでは使用できませんが、英語版BRIEFもセット販売いたしております。●お急ぎの方、上記以外のメディア希望の方、英語BRIEFのセット販売希望の方はユーザー・サポート係までご連絡下さい。

日本システム株式会社

●お問合せは　東京都立川市曙町1-18-2一清ビル立川事業所Briefユーザー・サポート係
TEL0425-23-4469　FAX0425-27-6127

〈資料請求番号004〉

タイアップ企画

ツール＆ライブラリ

2月8日発売

米国「The C Users Journal」「Windows™/DOS DEVELOPER'S JOURNAL」提携誌

月刊 Cジャーナル

C言語を主体とするコンピュータプログラマのための月刊情報誌

C JOURNAL

C JOURNALは、毎月8日全国の書店でお求めになれます。定価980円(本体951円)

3

MAR.1992

特集

デバッギング〔後編〕

■Cでのデバッグ―その概要―
■ヒープ関数を包装するデバッグ装置
■Cプログラムのための関数呼び出しツリーの作成
■スタックを吹ッ飛ばしてはいけません
■汎用コマンド行スイッチ

特別記事

▶D.ThielenのMemorandum

連載

●STANDARD C
●Dr. C's Pointers
●Questions & Answers
●Programming Style
●MINIX通信
●GUI環境
●プログラマの妻たち
●BART EXPRESS
●GNU MS-DOSへの移植
●Pascal言語系の制作
●たのむぜ！DOS/V！

付録 DISK

VZ-Editor Ver.1.57差分ファイル
Gnu gas
掲載記事の全コード

次号特集(4月号／3月8日発売)

ポータビリティ

株式会社ビレッジセンター 出版局

〒101 東京都千代田区三崎町3―1―16 神田アメレックスビル2F
●営業部:03-3221-1825●編集部:03-3239-1950

あのVisual Basicに、待望の解説書がいよいよ登場！

98シリーズで使う 本文2色刷

Visual Basic入門

好評発売中

A5判 208頁 定価980円(本体951円)

塚本吉彦著

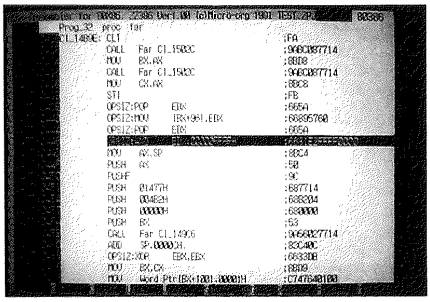
本書の内容

第1章 Visual Basicとは
[1] Visual Basicとはどんな言語だろう
[2] PC 9800シリーズで使うには
[3] Visual Basicの画面構成を覚えよう
[4] Visual Basicの操作を覚えよう
第2章 WindowsとVisual Basic
[1] Windowsのしくみ
[2] プログラムの処理
第3章 Visual Basicの約束ごと
[1] プログラミングの基本
[2] 変数・定数・演算子
[3] 制御構造
第4章 関数・ステートメント
[1] 配列の操作
[2] 型の宣言
[3] 画面の描画
[4] プリンタへの出力
[5] 文字列の出力に関する関数
[6] ファイルの操作
■付録1 プログラム例
(1) 遡って数値を確認できる足し算専用電卓／(2) 見積・納品・請求書作成プログラム／(3) DDEの利用例
■付録2 用語解説一覧

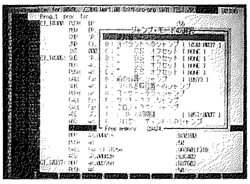
ご注文・お買い求めは、お近くの書店まで

発売：株式会社ビレッジセンター 出版局

〒101 東京都千代田区三崎町3―1―16 神田アメレックスビル2F
●営業部:03-3221-1825●編集部:03-3239-1950●FAX:03-3221-1768



ZZ386 最強 リバース・ツール



ジャンプ

絶対値：指定されたアドレスへ。
実効オペランド：オペランドの実効的なアドレスへ。
各セグメントオフセット：オペランドの値＋セグメント値へ。
ラベル参照位置：カーソル行のラベルの参照位置へ。
各タグへのジャンプ：任意のユーザ、サーチ、クロスリファレンスのタグへ。
CS IP の初期値：プログラムのスタート位置へ。
リロケートポイント：任意のリロレーションデータの行へ(EXEファイル)。
バックジャンプ：ジャンプのヒストリーを逆にジャンプします。

サーチ

ジャンプアドレス：指定されたアドレスへのJMP、CALL命令を探します。
ダイレクトアドレス：指定されたメモリーアドレスのMOV命令などを探します。
数値：指定された数値を操作している行を探します。
特定の命令：FAR、NEAR、メモリー参照ジャンプ、I/O、INTなど特定の命令行を探します。
文字列関連：文字列の行、マッチする文字列、マッチするデータ列を探します。
セグメント関連：全ての、または指定されたセグメントレジスタを変更している行を探します。

クロスリファレンス

ラベルデータをもとにクロスリファレンスを自動作成します。同時に未定義ラベル、未使用ラベルのデータも作成されます。これらのデータはいつでも画面上で確認でき、解析のヒントになるでしょう。またジャンプ機能から参照されます。ファイル、プリンターに出力できます。

マニュアルモードの操作性

ラベルはZZ386が自動生成する以外にユーザーが任意の名前で作成、リネームできます。コメントはZZ386が自動生成する以外にユーザーが書き込めます。命令、データ(バイト、ワード、DUPなど)、文字列などの属性はスクリーン上から行、ブロック単位で指定変更できます。

その他の機能

プロジェクトファイル（逆アセンブル諸条件、動作環境の設定）のメニュー作成。テキストファイルの表示、ファイルのダンプ表示。子プロセスの実行。EXEヘッダの表示。

- □強力になった自動解析能力とスクリーンエディター並みの操作で最高の環境を提供いたします。
- □EMS、GVRAM、ディスクスワップ、メモリーデータのガベージにより大容量ファイルに対応。
- □豊富なサーチ、ジャンプ機能は逆アセンブルを強力に支援いたします。
- □ラベル、コメント、プロシジャ、セグメントが自動作成されます。
- □逆アセンブルにより得られたソースファイルの全部または1部をファイル、プリンターに出力できます。
- □各コマンドの使い方はオンラインヘルプにより適時に確認することができます。
- □作業途中のデータはファイルに記憶され、そこから再開できます。

対応CPU：8086～80486
動作機種：PC9801シリーズ、PC286/386シリーズ、IBM PC/AT. DOS/V J4.0
（5"/3.5"デュアルメディア）
価格（税別）
¥39,000

ビレッジセンター、T-ZONE、システムロード、その他ソフトショップにて好評発売中です。
直接購入ご希望の方は銀行振込みまたは現金書留にてお願いいたします。送料は当社で負担いたします。郵パックの代引でも御注文承っております。
●振込口座：太陽神戸三井銀行　神宮前支店　普通 086-3056895

※広告に記載されている各製品の小数点以下のマイナーバージョンアップについては、無償で行なっています。
ご希望の方は、返信用封筒と175円切手2枚を同封の上、お申し込み下さい。
（現在の最新バージョンは、ZZ386＝1.09、ZASM＝3.09、ZZ＝2.06です。）

micro-org
株式会社 マイクロ・オーグ
〒164 東京都中野区南台1丁目1番14号サンハイツ208
PHONE：03-3375-3306 FAX：03-3373-6605
営業時間：月～金 AM11:00～PM6:00

〈資料請求番号005〉

本格派Cプログラマを目指す、すべての初心者へ!!

好評発売中

MS-C入門

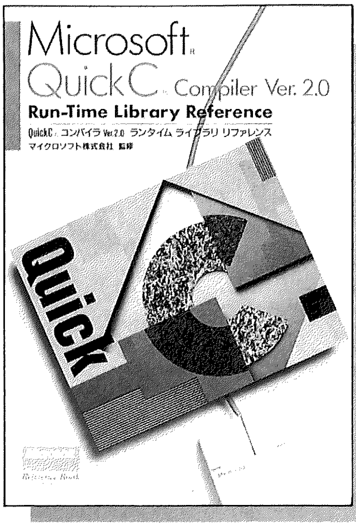
●定価2,600円　A5判・380頁
吉川敏則　著

本書は、初心者を対象にしたものですが、
初心者が初級のままとどまるのではなく、
中級のレベルに到達することを目標としています。
これまでC言語の入り口で挫折していた人も、
詳細に書かれたこの一冊を手に入れることにより、
Cプログラマへの道が開かれます。

SE Shoeisha Co., Ltd. 株式会社翔泳社 〒102東京都千代田区平河町2-4-14平河町KSビル TEL.03-3263-0447(代表)
翔泳社の本は全国どの書店でもお求めになれます。店頭にない場合は「至急」とお申し出の上ご注文いただければ10日位でお届けできます。

プログラミングのための優良図書

SE
Shoeisha Co., Ltd.



Quick C コンパイラ Ver.2.0 ランタイムライブラリリファレンス

Quick C Compiler Ver.2.0
Run-Time Library Reference

マイクロソフト株式会社　監修
A5判/364頁
定価2,400円

『言語ソフトのベストセラー「Microsoft Quick C」のオンラインヘルプを完全書籍化。マイクロソフト社監修によるパーフェクトQuick C リファレンス。プログラミング時の参考書として、また学習書として活用できます。』

Quick BASIC Ver.4.5 ステートメント&関数リファレンス

Quick BASIC Ver.4.5
Statement & Function Reference

マイクロソフト株式会社　監修
A5判/394頁
定価2,400円

『Quick BASIC Ver.4.5のQBアドバイザを完全書籍化。パソコンの前を離れて参照したい時など学習用として最適です。各ステートメント & 関数にサンプルプログラミングを満載したパーフェクトリファレンス。マイクロソフト社監修。』

Quick BASIC スタイルブック

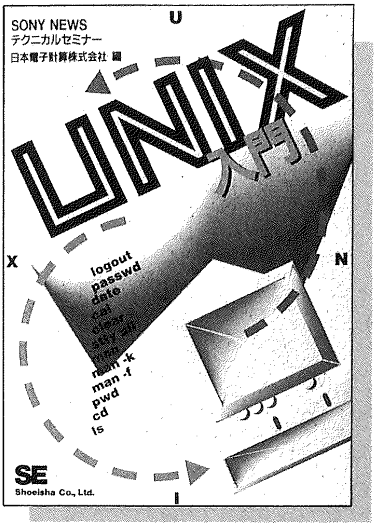
イント・ツーワン　著
B5判変型/236頁
定価2,300円

『初心者からマニアまで幅広く使われるQuick BASIC。このQuick BASICを用いて質の高いプログラミングを書くためのノウハウがいっぱいつまっています。一問一答の問題集形式。パズルのようにプログラミングが学べます。』

UNIX入門
―SONY NEWSテクニカルセミナー―

日本電子計算株式会社　編
A5判/237頁
定価1,800円

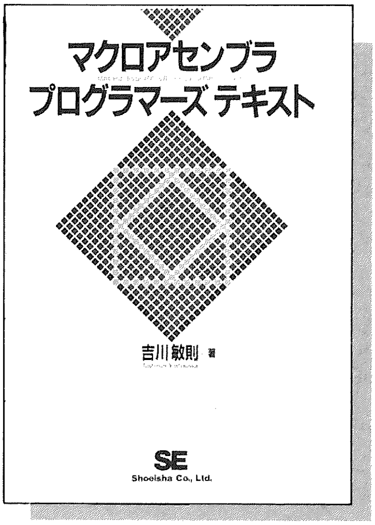
『ソニーのNEWSによる「UNIX入門」セミナーのテキストをベースとした入門書。UNIXのコマンド、エディタの使い方、日本語入力の方法、ネットワークの利用方法などを豊富な実例をまじえて解説。はじめてのワークステーションを利用するユーザ必携の書。』



マクロアセンブラ プログラマーズテキスト

吉川敏則　著
A5判/243頁
定価2,600円

『最低限必要なハードとソフトの知識から、データ演算の仕組みを踏まえた上で、アセンブラプログラミングの基本を丁寧に解説。アセンブリ言語に興味を持つ人や、これからアセンブリ言語でプログラムを作成する人に役立つ一冊。』



C文解釈
―「読む」ことからはじめるC言語―

豊国永健　著
B5判/183頁
定価2,000円

『C言語の文法書ではあまり取り上げられてこなかったわかりづらいポイントを、実際の応用プログラムを用いて解説。C言語の入門書とプログラムリスト中心の本との橋渡しになります。』

SE Shoeisha Co., Ltd. 株式会社翔泳社 〒102東京都千代田区平河町2−4−14平河町KSビル TEL.03−3263−0447(代表)
翔泳社の本は全国どの書店でもお求めになれます。店頭にない場合は「至急」とお申し出の上ご注文いただければ10日位でお届けできます。

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

特集
Cじゃなくてもいいじゃない
C言語を考え，プログラミングへの理解を深めるために

ANtI_C = ??-(FORTRAN & LISP & ALGOL);
C言語シンドローム　市川新　2

構造化プログラミングを具体化した教育用言語
Pascal vs. C　虎岩登　13

チームプログラミングに最適の言語仕様
MODULA-2 vs. C　株式会社　NJK　鈴木朋夫　24

MODULA-2でアプリケーションプログラムを開発する　36
●インタビュー　(株)アドミラルシステム　堀正明氏に聞く

究極のマクロ言語 Lisp
Lisp vs. C　生越昌己　40

論理プログラミングを実現する人工知能言語
Prolog vs. C　工藤栄太　56

Smalltalk-80とインヘリタンス
Smalltalk vs. C　神林靖　64

パソコン用の言語処理系　74

プログラム「遊」工房　拾四乃巻
bedit／簡易バイナリエディタ　谷口典生　80

IC千夜一夜　第9回
ビット・スライス型マイクロプロセッサとマイクロプログラム—その2—　虎岩登　86

イカすBASICプログラミング入門　第3回
謎のプロシージャ編　酒井法雄　91

実践C塾　アルゴリズム篇　第3回
サーチ・その1　加野島英渡・山本博昭　98

98活用アセンブラ技法　SCSIドライバ編　第3回
ディスクイメージダンププログラムの作成　星野操　110

付録
5”2HD フロッピーディスク

綴じ込み付録
MS-DOS Version 5.0 ファンクションリクエスト一覧

表紙イラスト：日暮修一
表紙 AD：佐々木幹夫(SE)

パソコン博物館　第15回
デスクトップパソコンの王様／日本電気　PC-9801　井戸祐光　76
プログラマの道具学　第15回
パソコン通信　祐安重夫　78
情報処理技術者試験　第２種合格ノトレーニング
第12回　問題と解答ー午後の部　121
Book Review「Scheme 入門」　中村正三郎　126
本のアンテナ　古田島義和
佐々木哲の U.S.プロダクツウォッチ　128
西川和久の希望的観測　130
コンピュータの法律　熊谷信太郎　132
アキバ短信
プログラマの仕事場　第12回　アルバソフト株式会社　134

INFORMATION PAGE　137
NEWS & NEW PRODUCTS　138
Programmer's Program PC-lint　142
オンラインソフト採集帳15「TRAINS」　144
HEALTHY? UNHEALTHY?
ドラゴン・ポスト　146
付録ディスクのご案内　148
バーチャルスペースガイド　149
読者プレゼント
クイズ・ランダムスクリーン　150
SE-NET からのお知らせ
バックナンバーのご案内　151

広告索引　資料請求番号

AX メーカー連合広告
(株) ライフボート……001
ボーランド (株)……002
富士ソフトウエア (株)……003
日本システム (株)……004
ビレッジセンター
(株) マイクロオーグ……005
T･ZONE……006
ボーランド (株)……P04

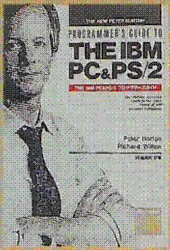
人材募集広告

(株) ソフトヴィジョン
(株) エルテックス
(株) ソフトビジョン
(掲載順)

1992 MARCH No.15

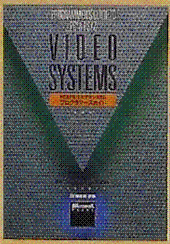
翔泳社が贈るプログラマ必携書

IBM PC & PS/2 プログラミング 3部作



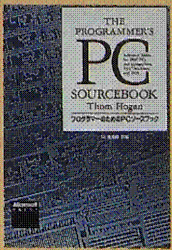
あのピーター・ノートンのベストセラー、IBM プログラマ必携のスタンダードガイダンス。

IBM PC & PS/2
プログラマーズガイド
● Peter Norton & Richard Wilton著
●定価　5,200円



IBM PC ファミリーのビデオシステム完全ガイド。

PC & PS/2 ビデオシステム
プログラマーズガイド
● Richard Wilton著
●定価　6,000円



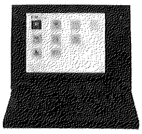
IBM PC & PS/2 プログラマ必携の技術資料完訳。

プログラマーのための
PC ソースブック
● Thom Hogan著
●定価　5,200円

IBM Authorized Dealer

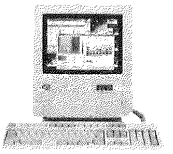
Net Ware取扱開始！ NEWマシンぞくぞく大量入荷!!

PS/55 note



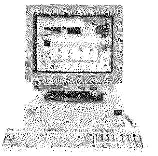
大容量80MBハードディスク内蔵（S18 SV8 S28 SA8）
パワフルなDOS/Vノート
80386SX(16MHz)2MB FDD内蔵

PS/55Z 30U



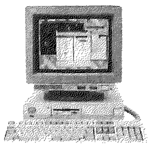
XGA採用パーソナルディスクトップ
80386SX(20MHz)2MB or 6MB
パソコンらしくないパソコン？オシャレ

PS/55Z 10S



386SX搭載の スタンダードPC
DOSV. VGA. ATバス採用
80386SX(20MHz)2MB or 6MB

PS 55Z 10T



PS/55Z 最強パフォーマンスPC
XGA高解像度・大容量
80386(20MHz)2MB or 6MB

ディスプレイディスパッチドライバー by C·F·Computing

今話題のS3ボードやTiGAボードに付属の英語ドライバーで日本語が使える!! **特価¥3,800**

日本語スーパーVGAドライバー

噂の800×600ドライバがとうとう出た
今すぐT·ZONEへ行こう！

800×600ドットのSuper VGA(S-VGA)を日本語対応にするスクリーンドライバ。

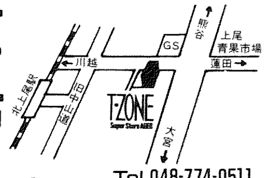
T·ZONEならでは **大特価¥2,500**

OPEN 2月29日(金)

北関東最大規模 コンピュータストアー誕生!!

新感覚のコンピュータショップ
オープニングセール
'92年2月29日～4月12日

T·ZONE Super Store AGEO



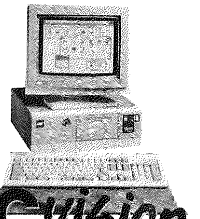
営業時間 AM10:30～PM7:00
〒362 埼玉県上尾市緑丘3-5-28
Tel.048-774-0511
Fax 048-774-0823

CREATIVELABS
マルチメディア対応のSOUND BLASTERは必需品です！ **特価提供中**
SOUND BLASTER

これ1つでマルチメディアが楽しめます。 内蔵CD-ROM SOUND BLASTER PRO MIDI KIT マルチメディアイクステンション他ソフト付 UPGRADE KIT ¥164,800	ステレオ22FM音源 CD-ROMインターフェイス他 SOUND BLASTER互換 Multimedia Sound Standard PRO ¥57,800
11FM音源 MIDIジョイスティックパワーアンプマイク入力 各種ソフト付 V2.0 ¥34,800	PS/55等のマイクロチャンネル版です。 MCA ¥64,800

AST RESEARCH JAPAN K.K.
ASTから DOS/Vマシン

新登場



80386SX/20MHz
FDDモデル ~~¥308,000~~
40MBモデル ~~¥355,000~~

- 高速32ビットCPUを採用。
- メインメモリ2MBを標準搭載。
- 80387数値演算プロセッサの搭載が可能。
- IBM PC/ATスロットを5基装備。

PCvision

マルチメディア対応高速CD-ROMドライブ

- 標準SCSI(最大7台まで同時接続可能)
- 高速アクセス(340ms)、64KBキャッシュ内蔵
- CD-ROM XA対応
- 市販オーディオディスク演奏可能

	ドライブのみ	AT I/F付	PS/2 I/F付
内蔵モデル	¥125,000	¥145,000	¥158,000
外付モデル	¥135,000	¥155,000	¥168,000

DOS/Vノートおすすめセット 限定

- CPU：80386SX/20MHz
- RAM：3MB
- FDD：3.5"×1
- HDD：40MB
- 外部VGA出力
- A4ファイルサイズ

＋ DOS J5.0/V ＋ シリアルマウス ＋ 一太郎dash

何と！何と！の超特価¥298,000

VGA

DOS/Vの必須アイテムはVGAアダプターです。
豊富な在庫から選べます。

BASIC VGA 640×480	¥19,800
SUPER-VGA 1024×768 16色	¥38,000
SUPER-VGA 1024×768 256色	¥38,000
Dyanbook用VGAアダプタ	¥49,800

VGAモニター

DOS/V対応やWINDOWS用として高解像大画面のモニターがほしいあなた、何を選びますか？

NANAO T660i（ケーブル別）
20"0.31mm 30～78kHz 1284×1024 ~~¥588,000~~

三菱 XC-98V3
14"0.28mm 98/VGA 640×460 ¥78,000

NEC MULTISYNC II
14"0.31mm 15.5-35kHz 800×560 ¥95,000

IBM 8515
14"0.28mm VGA. 8514/A. XGA ~~¥160,000~~

the Manhole（日本語版）

それはマンホールからはじまった……
ゲームを超えたファンタジーアドベンチャー発売中！

DOS/V対応
IBM PS/55
テラドライブ対応
¥9,800

DOS/V

Mitac 3060D

106キーセット 超特価¥270,000
101キーセット 超特価¥258,000

共通仕様
80386SX/16MHz、3.5"1.44MB×1、HD 40MB、スーパーVGA(512KB)、スーパーVGAモニター

GAME

時代はゲーム!? TERADRIVEや新作のソフトが山盛りでご来店をお待ちいたしております。

WALT DISNEYの永遠のアイドル、ミッキーマウスがあなたにしゃべりかけます。

MICKEY'S ABC	¥10,800
MICKEY'S 123	¥10,800
MICKEY'S ABC & SOUND SOURCE	¥13,800
SOUND SOURCE(驚異のローコスト音源スピーカーです)	¥6,800
JOYSTICK	¥6,800
ゲームカード(PC/AT)	¥9,800
ゲームカード(MCA)	¥15,000

80486/33MHz made in USA

超特価¥498,000

80486/33MHz
RAM 8MB(max 32MB)
128KBキャッシュ(max 256KB)
3.5"1.44MB×1/5"1.2MB×1
200MB IDE HDD
スーパーVGA ET-4000 W/1MB
2シリアル、1パラレル、1ゲーム
拡張スロット×8
ドライブベイ×6
ミニタワー、2ボタンマウス
106キーボード

80386/33MHz ノート made in TAIWAN

限定20台 超特価¥438,000

最強ノート

80386/33MHz
RAM 2MB(max 16MB)
32KBキャッシュ
3.5"1.44MB×1
40MB HDD
32階調VGAバックライト液晶
外部VGA出力
1シリアル、1パラレル
外部テンキー付属
外部101キーボード、外部FDDポート
ニッカドバッテリー(充電2時間、2時間使用)
280W×220D×54H 4kg

80386/33MHz セット

超特価¥398,000

RAM 4MB
64KBキャッシュ
3.5"×1、5"×1
80MB HDD
2シリアル、1パラレル
1ゲーム
マウス
101キー
スーパーVGA 1MB SVGAモニター

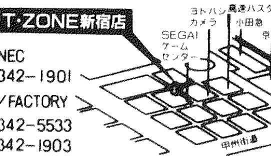
Microsoft BALLPOINT MOUSE 超特価¥28,000

VGA信号をビデオに変換するコンバーター 超特価¥98,000
320×200～640×480の信号をS-VHS、ビデオに変換して、テレビやビデオに接続できます。

MINI-SCSI 超特価¥29,800
プリンタポートにつなぐだけでSCSIデバイスが使えるようになります。

100MB HDDセット 超特価¥138,000

T·ZONE 新宿店 ☎03 3342-1901

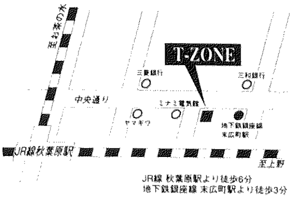


1F APPLE & NEC 03-3342-1901
2F IBM PS/FACTORY 03-3342-5533
FAX 03-3342-1903
〒160 東京都新宿区西新宿1-12-18

T·ZONE PS PLAZA IBMゾーン

3F ハードウェア
4F ソフトウェア
☎03-3257-2652
FAX 03-3257-2661

営業時間 AM10:30～PM7:00
株式会社 亜土電子工業



〒101 東京都千代田区外神田4-4-1

T·ZONEスタッフ募集 IBM、Apple、無線のスタッフを募集中です。アルバイトも募集中！ ☎03-3257-2690

☆T·ZONEの7Fは専門書籍(国内・外)・データーブックの専門フロアです。大量在庫中！ ほしい本は何でも揃います。

〈資料請求番号006〉

C言語を考え、
プログラミングへの
理解を深めるために

特集

Cじゃなくてもいいじゃない

世の中プログラミング言語といえばCである。猫も杓子も，プロフェッショナルもアマチュアも……，まるでC以外にはプログラミング言語が存在しないかのような勢いだ．たしかにCはなんでもできる便利なプログラミング言語だ．しかし，なんでもできるとは，裏を返せばなにもできないということではないのか？　また，思考を言語が規定するとすれば，Cしか知らないプログラマはプログラムを一面的にしかとらえられなくなるのではないだろうか？
世の中に蔓延しているCについて，その実体をとらえなおし，さらにほかの特徴あるプログラミング言語と比較することでプログラミング言語とはなにかについて考えてみたい．

特集
Cじゃなくてもいいじゃない

流通経済大学経済学部助教授
市川　新

ANtI_C=??-(FORTRAN & LISP & ALGOL)

C言語シンドローム

私たちは高水準プログラミング言語を求めている．有史以来，人間の知的活動が継承発展させてきた知識量はついに自己に備わった情報処理資源の枠を越えてしまい，今や，「考える石」を思考の道具として使いこなさなければならない．そのためには，「考える葦」と「考える石」の間には共通の言葉がなければならない．しかし，世の中には葦のために都合のよい言語と石のために都合のよい言語があるようだ．第5世代コンピュータの挑戦は葦のために都合のよい言語のみならず石自体を葦にする試みでもある．プログラミングが知的生産になるのは「葦の言語」での思考であり，「石の言語」での命令ではない．それはおそらくANtI_C言語であろう．

マシン語プログラミング

図1 マシン語プログラミング

```
>symdeb                                         デバッガの起動
-a100
3BC8:0100 mov dx,0000                           ①0000は仮の番地データ
3BC8:0103 mov ah,09                               文字列出力命令
3BC8:0105 int 21
3BC8:0107 mov al,00
3BC8:0109 mov ah,4c                               プログラム終了命令
3BC8:010B int 21
3BC8:010D db 0d,0a,'hello, world',0d,0a,'$'     ②出力データ
3BC8:011E
-a100
3BC8:0100 mov dx,010d                           ③0000を出力データ番地に変更
3BC8:0103
-g                                                プログラム実行

hello, world                                    ④実行結果

>                                                 コマンド・インタプリタへ復帰
```

プログラミング言語には，マシン語，アセンブラ語，コンパイラ言語という分類基準がある．インタプリタ型といわれる言語があるが，これは必ずしもコンパイラ言語と相対するものではない．これらは，言語の文法的仕様を意味するのではなく，ソースプログラムの記述内容をコンピュータに仕事させる方式についての分類である．したがって，インタプリタ型アセンブラ語もありうる．基本的にすべてのプログラミング言語はその属性として非インタプリタ型かインタプリタ型かの属性値をもっている．その区別は，人間が記述したプログラムを一括して処理するか，人間がプログラムを記述しているのと同時進行して部分的に処理を行うかの違いである．

一般に，コンパイラは，ソースプログラムを最適化技術により，無駄の少ないマシン語に翻訳するので，そのオブジェクトプログラムが実行されると，ソースプログラムに表現された仕事が効率的に処理される．インタプリタ型はソースプログラムの個々の文を読みながら，それに対応したより汎用的に記述されたルーチンを呼び出し，処理を進めていくから，実行の効率が悪くなる．しかし，プログラムを記述中のケアレスミスは実時間で修正できるから，プログラミングという作業は効率的になる．

ところで，DOS付属のデバッガを使えばインタプリタの感覚でマシン語が記述できる．これらのデバッガでは命令などに符丁が使えるのでアセンブラ語風マシン語になる．実際にマシン語でプログラムを記述すると8進表記か16進表記か，ときには2進表記することになる．図1ではデバッガで簡単なプログラムをマシン語で記述している．このプログラムはDOSの文字列出力システムコールを使ってコンソール画面に挨拶文を表示しようとしている．①の0000は画面に表示したい文字列データの先頭番地を示さなければならないが，その番地がこの行を記述しているときに確定していないために，番地を保存する領域を確保するための仮の番地データである．したがって，0000でなくてもよく，桁数が2バイトに相当すればよい．②の出力データを記述するときにその格納先頭番地が010dであることがはじめてわかる．そこで，③では，①で確保してある領域に確定したデータの先頭番地を格納し直している．このプログラムを実行すれば④のように挨拶文の文字列が表示されて，目的の仕事がコンピュータで処理されたことがわかる．

この例では，データの番地が当初未確定であるが，もちろん，ジャンプ命令で後方へ分岐するときもその命令の番地が未確定である．マシン語でプログラムを修正するということはこれらの番地も修正しなければならない．アセンブラ語ではこの番地の確定作業をコンピュータに代替させることができる．

DOSのアセンブラ語で図1と同じ仕事をするプログラムを記述してみたのが図2のプログラムで，①のdataは画面に表示したい文字列データの先頭番地を示す識別子である．その番地がこの行を記述しているときに確定している必要はない．その番地を保存する領域を確保することが指示されていればよい．②の出力データを記述するときにその格納先頭番地を示す識別子をdataとして命名している．アセンブラと呼ばれるプログラムのすべき仕事は命令などをビット情報に変換することだが，それより重要なことは番地の確定である．アセンブラ語では，プログラムを修正しても，そのために発生する番地の修正はコンピュータに代替されるので，プログラマの負荷増にならない．

ところで，図1および図2ともに，

番地データを必要としたときにはその番地が未確定であった．ところが，挨拶文文字列データがプログラムの先頭部にあれば事情が変わってくる．また，ジャンプ命令で自分自身よりも前方へ分岐すると，その番地は既知であるから確定している．しかし，プログラムを修正すればやはりすべての番地の確定を行わねばならない．この確定作業をコンピュータに代替させ，代替作業の効率化を追求すると，番地を参照する命令などでその参照番地が自分自身よりも前方にあればよい．参照したい番地が後方にあれば，それらの番地情報をあらかじめが用意するという作業が必要になる．つまり，その分だけ効率が悪くなる．

脇道にそれるが，プログラム中で番地情報が必要なときに前方参照という表現がある．この前方とは，ある命令文などを基準にして，参照したい番地が自分自身より後方ないし下の行に記述されているときに使われる．プログラムを記述している時点で，この行為に命名すると，先行参照という用語が望ましいと思われる．上述の文中でもこの立場で語用をしている．たとえば，図３では②で文字列の配列名を③の実体の定義より先行して参照している．そのために①の宣言において翻訳に必要な情報をあらかじめ付与している．データの実体が必要なのは実行時である．この種の先行宣言の方法はアセンブラ語を含めてほとんどの現代的言語に採用されている．COBOLでは構文上先行参照がまったくできないようになっている．Pascal系も先行参照ができない．

プログラミングの効率化の出発点はこのように番地に関する煩わしさからの解放にあったといえる．番地の確定に関する作業をコンピュータに代替させるというアイデアは現代的プログラミング言語の核になろう．また，コンピュータにその作業を代替させるときに，先行参照をしないという約束をプログラマが守れば，その作業を行うプログラムも効率化されることになる．

FORTRAN & LISP

FORTRANとLISPには共通したプログラミング言語の歴史がある．数多くあるプログラミング言語のなかで，これらの言語は伝統を誇るとともに，その後の拡張の努力により現役でありかつ新鮮である．個々のプログラミング言語の有用性を理解するには歴史をふり返ることが確実であり，現状から近未来への技術的動向の流れを読むことになる．

それでは，アセンブラ語とコンパイラ言語との違いはどこにあるのか．マシン語ではソースプログラムはそのままでオブジェクトプログラムである．アセンブラ語では，ソースプログラムはアセンブラ方式でオブジェクトプログラムに変換される．コンパイラ言語では，ソースプログラムはコンパイラ方式でオブジェクトプログラムに翻訳される．アセンブラ語で記述された命令は基本的に一対一に対応したマシン語の命令に変換されるが，コンパイラ言語で記述された命令は基本的に一対複数に対応したマシン語の命令になる．必ずしも正確な数値ではないが，マシン語で1000行のソースプログラムは，アセンブラ語ではやはり1100行のソースプログラムであるが，コンパイラ語では100行のソースプログラムにな

図２　アセンブラ語プログラミング

```
code    segment
        org 100
start:  mov dx,data                           ①dataは出力データ識別子
        mov ah,09
        int 21                                  文字列出力命令
        mov al,00
        mov ah,4c
        int 21                                  プログラム終了命令
data    db 0d,0a,'hello, world',0d,0a,'$'     ②出力データ
code    ends
        end start
```

図３　先行参照と先行宣言

```
include <stdio.h>
extern char msg[];                            ①先行宣言
void main()
{
    printf(msg);                              ②先行参照
}
char msg[] = "hello, world¥n";                ③被参照変数
```


PC-9801 I/Oポートマップ
①はPC-9801/E/F1/F2/F3/M2/M3（おもにCPUが8086の機種）

るであろう．

図4では，コンパイラ言語命令と翻訳された命令群(マシン語ではわかりにくいのでアセンブラ語で表現)との関係を例示している．もちろん，TATE, YOKO, MENSEKIの領域と必要なデータは用意されているものとする．

①の表現はローマ字をはじめて習う小学5年生でも，それ以前に面積の知識を学習しているので，その意味するところは理解できよう．②のアセンブラ語の表現はアセンブラ語について相当の知識がないとその意味がわからない．つまり，コンパイラ言語の目的は，一般に，コンピュータが発明されるよりも以前から人類が蓄積し継承してきた知識体系とその表現を反映させたプログラム記述を可能にすることにある．とくに既存の知識との調和を意識したプログラミング言語を高水準言語という．もちろん，高水準言語で記述されたプログラムの仕事をインタプリタ方式で実行することもある．業界標準BASICはその代表例である．さらに重要なことは，副次的にハードウェアに依存するマシン語命令の表現の違いを吸収し，ソースプログラムの共通性により，異なるハードウェアでの処理を可能にすることである．

歴史的にみると，高水準の程度が既存の知識体系と整合するほどその寿命は長い．もちろん，その知識を獲得した専門家が多いほどその傾向がある．この意味で，最古の高水準プログラミング言語は，よく知られているように，FORTRANである．すでに，私たちは母国語を含めて多種の自然言語を創造的に継承してきている．さらに，ある特定の目的のために，人工言語を定義した．お互いが共通の非母国語で対話しようという目的で作られたエスペラント語がよく知られている．対象の表現で曖昧な部分ができないように，また，その表現形式の操作性に柔軟であることを目的としているのが，数学の表記法と化学の表記法であろう．コンピュータは，もともと計算する機械として発明されたのであるから，数学の知識ないし表記を反映した高水準プログラミング言語が研究開発されるのは自然である．

図4 コンパイラ言語命令と翻訳命令群

```
        INTEGER MENSEKI, TATE, YOKO

        MENSEKI = TATE * YOKO              ①FORTRAN命令


TATE    dw    ?
YOKO    dw    ?
MENSEKI dw    ?
        dw    ?

        mov   ax,TATE                      ②アセンブラ命令群
        mov   bx,YOKO
        mul   bx
        mov   MENSEKI,ax
        mov   MENSEKI + 2,dx

    int menseki, tate, yoko;

    menseki = tate * yoko;                 ③C命令
```

FORTRANは1955年初期から開発されて1958年から広く使われるようになりはじめた．現在でもFORTRAN90として現役であり，数値計算の領域ではほかに競争相手がいない．この頃，もう1つの重要なプログラミング言語が研究開発された．場所は後にタイムシェアリングシステムとBASICが誕生することになるダートマス大学で，人工知能を主題とした会議が開催された．この会議がきっかけとなり高水準リスト処理言語の開発が始まった．そのときに，現代的表現をすれば，プロトタイピングの環境として開発されたばかりのFORTRANの機能が援用された．さらにいい換えれば，当時唯一の高水準言語であるFORTRANでシステムプログラムが記述されたことになる．

LISPは1958年末から開発されて，1960年から広く使われるようになりはじめた．この言語は高水準言語であるが，人工知能プログラミングでは，PROLOGが高水準言語とされ，その比較からAI向けアセンブラ語といわれる．それにもかかわらず，やはり，記号処理を目的とする高水準言語である．LISP語の表記はそのままの形式で人間にとっても記号処理の表記形式になる．ところが，LISPは文字ないし記号処理向けであって，数値計算向けではない．コンピュータの構造は数値計算向けにできていても記号処理向けにはできていない．FORTRANの対象はハードウェアの処理方式と効率的に対応するが，LISPの対象はまったく対応しない．たとえば，LISPでは論理的にメモリは無限にあるものとしてプログラムが記述される．そこで，もっとも柔軟なデータ構造であるリスト構造をソフトウェアで実現し，その上に被せる形でLISPの機能を提供している．

そこで，図5に示すように，LISPに

はリストの分解と生成の関数が基本関数として提供されている．というより，すべてのデータとプロクラムが区別なく，リスト構造で一元管理されており，データ(副作用のないもの)であるかプログラム(副作用があるもの)であるかは，それが評価対象になった時点で関数形式になっていればプログラムであり，そうでなければデータである．つまり，データであるかプログラムであるかを決定する機構をプログラム自体に含めることができる．したがって，①のようにリストを多重にかつ自由に参照しその構造を動的に変更できる．②はリストの多重間接とほぼ同じ内容をC言語の間接演算子で表現した．もちろん，関数もC言語ではポインタとして間接参照できる．しかし，それを混在することができなくなりつつある．

面白いことに，プログラム内蔵方式はメモリ上の内容がプログラムであるかデータであるかの区別はしていない．プログラムカウンタにより参照されたものが命令になるだけである．LISPの関数評価はハードウェアの命令評価と似ている．データであるかプログラムであるかは表裏の関係であるといえよう．

いずれにせよ，人間の知的活動の観点からすると，FORTRANがハードウェアの呪文に縛られて創造された言語とすれば，LISPはハードウェアの特性をソフトウェアで隠した上で，自由に創造された言語といえよう．

LISP & ALGOL

LISPは関数型言語といわれる．それに対してFORTRANは逐次型言語といわれる．もちろん，アセンブラ語も逐次型言語である．後述するがPROLOGは論理型言語である．

また，横道にそれるが，プログラミング言語の属性には，逐次型，関数型，論理型の区別がある．逐次型言語というかわりに，手続き型言語ということがある．これも後述するがいささか誤解を招く語用である．プログラミングスタイルという観点からすると，ある表現したいことが手続き型の知識であれば，現在のところ，3通りの表現が考えられる．すなわち，手続きを処理単位の実行文系列として表現するものを逐次型言語，処理単位の式の値の入出力依存で表現するものを関数型言語，処理単位に述語の論理値の関係で表現するものを論理型言語という．LISPは関数型言語になる．ただし，実用の処理系では，論理型言語でも逐次型の形式を一部採用しているように，相互乗り入れの部分がある．

命令文が値を返すことを前提にすれば，プログラムの表現法を関数形式に統一できる．図6で比較してみる．①のFORTRANの命令表現で条件は値を返さない．それを②のように条件の作用をする関数が表現できれば簡潔になる．それを発展させ，③のようにそれぞれが値をもつ関数形式で表現できるようにしたのがLISPの革新であろう．関数表現すると個々の関数の独立性が強くなる．さらに重要なことは再帰呼び出しが可能になることである．自分で自分自身を呼び出せると，ある種の仕事を洗練された表現で記述することができる．しかし，逐次型言語はハードウェアの構造を素直に反映しているが，関数型言語および論理型言語はハードウェアの構造と衝突するものである．つまり，どちらの言語も再帰呼び出しが可能であるが，それは逐次型ハードウェア向きに変換しているのであり，その方法はスタックで具現化している．スタックはメモリの消費が動的であるためにその必要量を特定しにくい．したがって，プログラムの記述という知的活動中には無限メモリの存在を前提にしている．④は，C言語の表現で値をもつ条件文では，唯一の3項演算子である．

ALGOLは，1958年から60年にかけて数学的意味での美しさを追求して創造されたプログラミング言語である．C言語の系譜は，C言語からB語へ，さらにCPL語へ，さらにALGOLに遡る．ALGOLの美しさは，ハードウ

図5 リスト処理とポインタ処理

①リスト分解関数

```
(SETQ ELEMENT (CAR (QUOTE (1 2 3 4) ) ) )
(SETQ ELENENT (CADR (QUOTE (1 2 3 4) ) ) )
(SETQ ELEMENT (CADDR (QUOTE (1 2 3 4) ) ) )
(SETQ ELEMENT (CDDDR (QUOTE (1 2 3 4) ) ) )
```

②間接演算子

```
int num, *car, **cadr, ***caddr, ****cddr;

num = *car;
num = **cadr;
num = ***caddr;
num = ****cdddr;
```

ェアの特性を隠蔽することとハードウェア特性そのものであるマシン語への翻訳効率とその実行効率を犠牲にしないという矛盾を両立させたことにあろう．C言語は実行効率を優先しながら継承してきた美しさを妥協できる美しさまで後退させていることになろう．この美しさの価値観は，PascalやMODULA-2などに積極的に継承されていることが知られている．業界標準BASICはこの美的感覚を許されない厳しいハードウェア環境で育成されてきた．しかし，ニューベーシックを含めて手続き的知識を対象としたプログラミング言語はハードウェア環境に余裕が生まれ，食足りてきたので美しさを構うようになってきている．もちろん，標準規格が尊重されるに従って，「C言語よおまえもか！」でその例外ではなくなりつつある．

さて，ALGOLは構造化プログラミングを可能にした言語仕様をもつ．ブロック構造を含む制御の3構造を取り入れ，その構造も種類が豊富である．プログラムの美しさの価値観はこのALGOLにはじまる．この言語の設計思想は，高水準言語の鏡となるもので，数学の標準的表記に準拠すること，可能な限り既得の知識でプログラムの内容が理解できること，プログラム自体がプログラムの内容を曖昧なく伝えること，コンパイラに負荷を与えずに効率的なマシン語を生成できること，である．C言語の浸透とともに何か大事なものを忘れはじめているように思えてならない．

図6 関数値の導入

```
X = ICAR(Y)                     ①ＩＦ命令
IF(N.EQ.0) X = ICDR(Y)

X = IF(N.EQ.0, ICAR(Y), ICDR(Y))   ②ＩＦ関数

(SETQ X (COND ((EQ N 0) (CAR Y)) ((T) (CDR Y)) ) )
                                ③ＬＩＳＰ条件文

x = n == 0 ? car(y) : cdr(y);   ④Ｃ言語３項演算子
```

図7 FORTRAN90の特徴

```
CHARACTER (LEN = 20):: NAME
CHARACTER (LEN = 20, KIND = KANJI):: NAMAE          ①整数型文字列

RECURSIVE FUNCTION FACT (N) RESULST (FACT-RTN)
  INTEGER, INTENT (IN) :: N, FACT-RTN
  IF(N <= 0) THEN
    FACT-RTN = 1
  ELSE
    FACT-RTN = N * FACT (N - 1)                     ②再帰呼出部
  END IF
END FUNCTION FACT

INTEGER, TARGET :: YY
INTEGER, POINTER :: MM, DD
YY = 1992
MM => YY                                            ③ポインタ処理
DD => MM
YY = 4
PRINT *, DD

TYPE KEYBUF                                         ④型宣言
  INTEGER :: LINE_NUM
  CHARACTER (LEN = BUF_LEN) :: BUF
  TYPE (KEYBUF), POINTER :: NEXTPTR
END TYPE KEYBUF

TYPE (KEYBUF) :: CONSOLE_LINE

INTEGER (KIND = 2) :: I, J, K                       ⑤内部表現指定
```

温故知新言語

すでに述べたように最古高水準言語はFORTRANであるが，その最新版は最近のプログラミング言語の潮流である3Kが実現されている．3Kとは「国際化・国際化・国際化」の頭文字で英語ではトリプルアイといえよう．第1の国際化として，そのプログラムの対象のデータが英語以外の言語系に対応していることがいえる．つまり，漢字のデータについてもプログラム記述で陽に表現できる．第2の国際化として，プログラミング言語間の国際化であり，他のプログラミング言語の文化と言葉を輸入していることがいえる．第3の国際化として，異なるハードウェア間での環境にカルチャーショックを受けないために，基本データの内部表現の差をソースプログラムで記述できるようにしている．

図7はFORTRAN90の3Kの代表的例を示している．プログラミング言

語は従来英語圏での利用しか考慮しておらず，たとえば日本語のデータのために，国内用として独自の拡張が行われていた．しかし，図7の①のようにボーダレスの時代とともに各国語の文字表現に国際標準として対応している．

②③④は，ソフトウェア工学としてその有用性が定着しているものを積極的に取り入れている．いままで，ある副プログラムが回り回って自分自身を呼び出すことは禁止されているか暴走するかのいずれかであったが，②のように陽に再帰関数であることを指示することによって，再帰呼び出しが可能になっている．また，ポインタについても，特殊演算子とともに実現されている．

③の例では，MMはYYの番地情報を代入され，MMとDDはともにYYの番地情報を代入されている．したがって，YYの値を変更するとその値を基にして間接参照されて，この場合は4と処理結果が表示される．また，従来，構造型データがまったく定義できなかったが，④のように任意の構造型データを定義できるようになっている．ここまでFORTRANも進化すると，数値計算向き言語と限定することができなくなる．

④はデータの内部表現をプログラム上で制御しようとしている．実は，FORTRANは昔から移植性のよい言語として知られている．コンパイラの処理系が異なっていても，ほとんど問題なくマシン語を生成する．しかし，データの内部表現はハードウェアに依存するので，精度の要求されるプログラムでは実行結果に一抹の不安があった．最新版では，その基本内部表現についてもそのビット数などをプログラムで指定したり，その精度情報をプログラムで取り出し，移植することを前提としたプログラムを記述できる．

宣言的知識と手続き的知識

わが国では「ブームが去った」と感じるのはPROLOGである．発祥の地のヨーロッパはもとより米国で，いまなお，PROLOGへの期待は大きい．すでに述べたように論理型プログラミング言語であり，宣言型知識との整合性がよい．

PROLOGの基本記述要素には，規則の表現形式，その形式の一部である事実の表現形式と質問の表現形式がある．人間の知識にはあるがままの存在を表す事実と事実の相互間の抽象的表現である規則があり，それらを宣言的知識という．図8の①は前者が事実と規則の例を記述している．後者および②③は階乗計算するプログラムであり，手続き的知識例として記述している．宣言的知識は論理型言語で表現できるが，関数型言語や逐次型言語では直接的には表現できない．

純C言語再考

さて，C言語の特徴について再確認してみよう．K&RによればC言語は万能プログラミング言語であるという．OSカーネル記述にアセンブラ語の代わりに使われ，そのOS上で機能する

図8 宣言的知識と手続き的知識

```
otoko(katsuo).
onna(sazae).
onna(fune).
fusai(masuo, sazae).
fusai(namihe, fune).
haha(sazae,fune).
haha(tarao,sazae).
chichi(Child, Father):-haha(Child, Mother),fusai(Father, Mother).


kaijo(0, 1):-!.
kaijo(X, A):-W1 is X - 1, kaijo(W1, W2), A is X * W2.
```
①ＰＲＯＬＯＧ表現

```
(DEFUN KAIJO (X A)
    (COND ((ZEROP A) 1)
        (T (TIMES M (KAIJO X (SUB1 A))))))
```
②ＬＩＳＰ表現

```
unsigned int k(unsigned int x)
{
     return x == 0 ? 1 : k(x - 1) * x;
}
```
③Ｃ言語表現

```
unsigned int k(unsigned int x)
{
     if( x == 0) return 1;
     return k(x - 1) * x;
}
```


PC-9801 I/Oポートマップ
⑤はPC-980U2/VF2/VM0/VM2/VM4/UV2/UV21（オプションボード装着時）

プログラムの大部分がC言語で記述されている．現在ではハードウェアの壁を越えて，ほかのOSでもCプログラミングの環境が整っている．本来システム記述言語であるが，より一般的な分野でも採用されている．

言語仕様では，比較的豊富なデータの型宣言が用意されている．基本データとして，文字型，整数型，浮動小数点型があり，それらはそれぞれ複数の精度の内部表現を選べる．さらに，配列，構造体，共用体，ポインタ変数とポインタ参照などやそれらによる新しい型宣言が利用できる．

制御の3構造もSWITCHやWHILEなど豊富であり，反復構造の強制反復と強制中断も可能である．ブロック構造とその入れ子構造，ブロック内で変数宣言も自由にできる．関数は再帰呼び出しが可能であり，任意のデータ構造を関数値にもたせることができる．

ポインタと構造体のないC言語は「クリープのないコーヒー」ともいわれる．しかし，すでに，それはC言語の問題ではなく，手続き的知識をプログラミングする言語に共通する概念である．

C言語ではソースプログラムにほかのソースプログラムを読み込む制御ができるが，よく思いだしてみれば，COBOLで標準規格として同様なことが昔から行われている．これはソースプログラムによるライブラリ管理である．

また，入出力命令や組み込み関数がないといわれる．正確には，あらかじめ，コンパイラに組み込まれていないということであろう．しかし，K&Rの著作を含めて，どのC言語の書籍も最初に取り上げるプログラム例は次のものである．

```
#include <stdio.h>

main()
 {
  printf(" hello, world¥n");
 }
```

たしかに，Cコンパイラはこの出力命令をマシン語に翻訳する知識をもたされていない．ただ単に，あらかじめ，C言語で記述されたprintf関数が翻訳され，ライブラリに登録されたものを呼び出し連結しているにすぎない．その関数の実行効率を高めるかプログラムサイズの大きさを小さくする必要があれば，関数におけるアセンブラ語記述の部分の比率を高めることになる．つまり，Cコンパイラは関数のテンプレートないし形式的表現のみを翻訳していることになる．これによって，コンパイラの負担は軽減される．ところが，標準ライブラリなるものがプログラマズニーズにより流布されてくると，結果的にprintf関数はC言語固有の関数と同じになる．組み込みの関数があるかないかは，ソースプログラムからオブジェクトプログラムへの翻訳技術上の問題に帰着する．

PROLOGでも，同じように，述語による規則の表現のテンプレートないし形式的表現のみが約束され，その述語の意味は自由にプログラマが定義することができる．また，LISPには純LISPという概念がある．純LISPを構成するLISP環境は，原始的機能，すなわち，LISP自身で拡張的に記述し実現できない機能のみを備えたLISP処理系である．もちろん，効率などが障害になれば，LISPで記述した内容をアセンブラで記述すればよい．つまり，Cコンパイラは，純C言語コンパイラであるといえる．

表1には純C言語と思われる形式的表現をまとめてみた．これまでに述べた内容と比較すると，プログラマからみたC言語の特徴は作用素表記が記号式であることがわかる．これは，アセンブラ語よりも意味的に冗長性のない危険な表記であることに気付かなければならない．それは，高水準言語の歴史に逆行しているといえまいか．効率的記述を優先した表記法である．

表1　純C言語

<table>
<tr><td>基本データ構造</td><td>文字，整数，配列，構造体，共用体，ポインタ変数など</td></tr>
<tr><td>基本副プログラム形式</td><td>関数テンプレート</td></tr>
<tr><td>基本副プログラム構造</td><td>再帰可，再入不可</td></tr>
<tr><td>制御構造</td><td>連続型，反復型，選択型，強制反復，強制中断など</td></tr>
<tr><td>参照法</td><td>先行定義</td></tr>
<tr><td>変数有効域</td><td>大域変数，入れ子局所変数</td></tr>
<tr><td>編集連係</td><td>ソース・プログラムおよびオブジェクト・プログラム</td></tr>
<tr><td>作用素表記</td><td>記号方式
( ) . [ ] ->
! ˜ - * & ++ --
* / %
+ -
<< >>
< <= > =>
== !=
& ˆ | && || ? :
+= == /= *= %= <<= >>= &= =
|=
その他</td></tr>
</table>

グレシャムウィルスの増殖

ところで，情報・知識・データを厳密に区別しなければならなくなった．コンピュータの世界ではビットという単位がよく使われる．もともと，シャノンの情報理論（通信理論が正しい）で定義されている．その意味を念のために確認しておこう．

送り手は受け手にある情報を送りたい．ビットという単位は，その情報を受け手が受領したときにその情報についての曖昧さがなくなるが，その曖昧さを示す測度である．このときに送り手から送られてくる可能性のある情報のすべてがあらかじめわかっていて，かつ，個々の情報の起こり得る確率もわかっていなければならない．

たとえば1ビットの情報量は，送り手と受け手の間に「西武ライオンズと広島カープのナイター試合があり，勝負は五分であるとし，試合を見ていた送り手が受け手に電話をして，ライオンズが勝ったらベルを1回，カープが勝ったら2回鳴らして電話をきる」という約束があるときに該当する．つまり，電話のベルを1回分余分に鳴らすか鳴らさないかが1ビットになる．これはベルの回数を知ればよいのだからこれをデータという．送り手と受け手の間にナイター試合が話題になったときにそれを情報という．

あるいはこの情報のことを意味的情報というときはデータのことを形式的情報という．情報理論の符号化とは意味的情報を形式的情報に変換することをいい，復号化とは形式的情報を意味的情報に戻すことをいう．知識とは個々の情報の相互関係をいう．つまり，プログラムの用語でいい替えれば，ある目的のためにどのような変数を必要とするかを考えているとき，それが情報になり，その変数の相互関係が知識であり，その変数の内容がデータになる．手続き型知識では変数の相互関係が順序に依存し，階層構造になれば宣言的知識といえる．

今回は触れなかったが，C++は手続き型知識を記述するだけだったCに宣言的知識も記述できるように拡張したものといえる．さらに，相互関係が一義的に決定していれば，従来のプログラミング技術の延長になるが，相互関係が情況に応じて決定するときは，人工知能のプログラミング技術が必要になる．ただし，その宣言的知識は，そのプログラムの一構成要素となる手続きを抽象化したものであり，プログラムの対象に関するものではない．その知識は，C言語を基準にしたものではなく，プログラマを基準にしたものである．抽象的プログラムに，プログラミングの過程で目標との差分情報を顕在化させながら，それを具体的プログラムにまで詳細化する意味で，++Cと呼ぶのがふさわしいと思う．そうすると，++Pascalでも++BASICでもよいことになる．つまり，具体的プログラムの水準では，その言語は別の基準で選ばれてよいことになる．

また，データにも四則演算が適用できるものと，順序ないし同じか否かの論理演算のみ適用できるものとがあり，前者を定量データ，後者を定性データという．コンピュータのハードウェアは確定した手続き型知識のプログラムで定量データを処理するときにもっとも効率がよい．一方，人間は情報の創造と知識化に優れた能力をもっている．シャノンの理論は形式的情報について適用でき，意味的情報については適用除外になる．一般にコンピュータの知識をもちはじめた人は「形式的情報=情報」と思いこみ，シャノンの情報理論に先行して広がっている意味的情報の世界を忘れてしまう．その原因はグレシャムウィルスの感染である．

グレシャムウィルスは形式的情報処理部に発病し，その悪性的進行が広がると意味的情報処理部に侵食することになる．このことをグレシャムの法則という．図9はプログラマ自身の情報処理資源をモデルにしたものである．このモデルは，個人だけではなく組織にも当てはまる．私たち自身の情報処理はある一定の大きさの資源しかない．それは，基本的には評価基準の異なる意味的情報処理系と形式的情報処理系で構成される．意味的情報処理は帰納

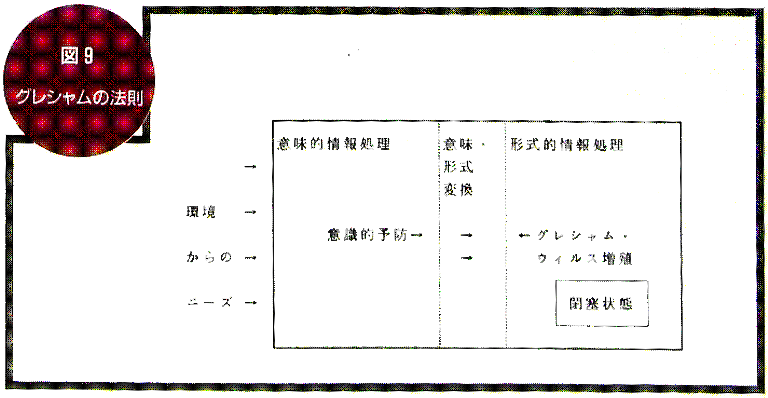


図9 グレシャムの法則

PC-9801 I/Oポートマップ
ポートアドレスについては一番右側からbit0〜bit15の順に表示されている．0はbit OFFで，1はbit ONを示す．

的推論に基づく情報処理であり，形式的情報処理は演繹的推論に基づく情報処理である．その中間に意味的情報を形式的情報に変換する部分があるが，いまのところよくわかっていない．あるいは分析と設計の関係といえば理解されよう．

目に見える具体的成果が期待できる私たちの生産活動はもっぱら形式的情報処理に依存している．一方，意味的情報処理は刻々と変わる環境からの要請を受け取らねばならない．それに何らかの変換を行い形式化することになる．その情報処理の方式は，意味的情報処理系では，より信頼性の高い情報処理方式を必要とし，形式的情報処理系は，より効率のよい情報処理系に置き換えねばならない．つまり，形式的情報処理の対象は，環境を源泉とし，意味的情報が形式的情報に変換されて，刻々と組み合わせ爆発的に増加することになる．この解決法は，より効率的な形式的情報処理方式に置き換えることしかない．コンピュータはそのための道具である．これを怠るとパンク状態になり，形式的情報処理の閉塞状態に陥る．そうすると，形式的情報処理は意味的情報処理の資源を流用しようとする．すべての情報処理資源がそのために消費されてしまう．

私たちは，資源の平衡的分配のために，意識的に予防を行うことか求められている．情報処理に関わる私たちのよき協力者は，形式的情報処理をもっとも得意としている機械である．グレシャムの法則をよく理解しておかねばならない．意識しなければ，私たちの情報処理資源は形式的情報のために使われてしまう．

C言語シンドロームの弊害

C言語シンドロームはグレシャムウィルスに感染した状態をいう．その診断基準は，「Cで，バンバン，プログラムを書いている」といえばまちがいなく陽性である．そのほかに，「Cのプログラムは読みやすい」，「Cのマニュアルなしでプログラムが書ける」と自慢する人も陽性である．「夜中にCでいろいろなユーテリティを作っているよ」という人も陽性である．「Cは便利で万能だと思うよ」という人も陽性である．

C言語で，バンバン，プログラミングできるには，記号とその作用に精通しなければならない．したがって，初心者には非常にとっつきにくい言語である．ところがある段階まで我慢をすると，急に親密な関係になる．ちょうど，グレシャムの法則により，プログラミングの作業が目に見える形で効率的になってきた段階である．これより先は，急激に脳の働きがC向けになってしまう．そうすると，まわりにあるものは，C向きの対象しかないように思えてくる．大事なそれ以外の事柄に気づかない．これをC言語シンドロームという．これはプログラマの病状である．

C言語の存在価値は，高水準言語ではなく，高水準アセンブラ語でもなく，高水準標準アセンブラ語にあると思っている．その意味では，マシン語ないし，アセンブラ語で記述できることはC言語で記述できないと困る．DOSのCであっても，マルチタスキング環境になれば，再入可能プログラムが記述できないと困る．OSカーネルが記述できるのであれば，デバイスドライバも記述したい．最近のANSIへの移行は，K&Rの時代から比較すると，どうも標準アセンブラ語ではなく，高水準手続き記述言語への指向に思える．たとえば，関数ポインタによる参照が記号表現からから語表現でもよいようになっている．これまでの，

```
(* function)();
```

から，

```
fucntion();
```

でも可能になった．構造体ポインタの参照法と同じように記号を使わなくなっている．また，列挙型が採用されたので，

```
enum fattri { VOL, DIR,
SYS, HIDEN, ARV, RDONLY }
ffalag;
```

という表記が可能になった．プログラムの可読性が向上している．プログラミング言語では英文形式のCOBOLが可読性のよい言語である．しかし，それは英語圏での話であって，日常の英単語を利用すると，日本語圏ではプログラム記述の作業に時間がかかるだけではなく，キータッチミスの可能性も高くなる．グレシャムの法則から推論すると，直感的に意味のわからない英単語を用語とするより，漢字のパターン認識と同じように記号を使うほうが苦痛がないかもしれない．昨今のCブームはこんなところに理由があるのかもしれない．一方，英語圏では，意

味のわかる語と無味感想な記号が混在すれば，当然，記号が嫌われてプログラムの楽な単語を利用するようになるのではなかろうか．つまり，PascalやMODULA-2への指向が強いことになる．いずれにせよ，Cプログラムには将来のために十分なコメント行を書き込んでおく必要があるが，

```
/*　コメント　*/
```

の表記法は手間がかかり，かつ見にくい．

他方，標準アセンブラであることを意識して，ユニバーサルポインタが利用できる．

```
void *
```

は，本来のC言語のあるべき姿と思う．エンディアンをプログラムで明示できることはFORTRAN90の例に見られるように，異なるハードウェアへの移植性を考慮すれば当然である．インテグラルプロモーションはいわば異なるC処理系での問題解決になるであろう．しかし，C言語がOSとOSまわりのシステム記述言語であるとすれば，グレシャムウィルスに完全に感染してほしい．高水準コンパイラ言語よりも標準アセンブラ語が求められている．

C言語シンドローム予防法

グレシャムウィルスの有効な治療法はいまのところ知られていない．もし，感染し陽性になると，おそらくプログラマの知的情報処理資源を使い果たし，情的情報処理資源を流用することなり，不幸なことになる．そこで，ウィルスに感染しないためにC言語シンドロームの予防法について考えてみる．

それは意味的情報の世界を忘れないことだ．ところどころにC言語の世界がいかに小さいかを述べた．それに対して，私たちの知識の世界は限りなく広く，そして，宇宙とともに刻々と膨張している．目の前には意味的情報処理の対象がゴロゴロしている．たしかにC言語は妖しい魅力的な言語である．しかし，C言語はONE-OF-THEMのプログラミング言語であるだけではなく，形式的情報処理系の最右翼に位置することを忘れてはならない．そして，その評価基準は効率であるので，形式的情報処理の閉塞状態を超越するために，近いうちに，さらに効率のよい新しい言語に置き換えられる．また，そうでなければ人間の創造的知的活動は停滞してしまう．C言語はマニュアルを参照しながらプログミングをしよう．Pascalの教えのとおり，私たちには「脳」を「石」にしない努力が求められている．

（注：プログラム例は特定の処理系で機能するものではない）

参考資料

1) Birners WJ(1989) High-Level Languages and Software Applications. McGraw-Hill

2) MacLennan BJ(1983) Principles of Programming Languages: Design, Evaluation and Implementation. Holt-Saunders.

3) Brainerd WS, et al.(1990) Programmers' Guide to FORTRAN 90. McGraw-Hill

4) Libes D, et al.(1989) Life with Unix. Prentice Hall

5) Kernighan BW, Ritchie DM(1988) The C. Prentice-Hall

6) ウィンストン, PH, 白井他訳(1982) LISP. 梅風館

7) 市川新 (1986) PROLOGシステム. 啓学出版

8) 北川敏雄 (1969) 情報学の論理. 講談社

9) サイモン, HA, 高宮訳(1977)システムの科学. ダイヤモンド社

10) 伊藤英治 (1987)「ANSI C言語標準規格案の詳細」インターフェース10月号

PC-9801 I/O ポートマップ
機種：1|2|3　ポートアドレス：XXXX00XX00000XA0　割り込みコントローラ（マスタ）　使用チップ：μPD8259A（8259相当）／μPD71059（8259相当）

特集 Cじゃなくてもいいじゃない

虎岩　登

構造化プログラミングを具体化した教育用言語

Pascal vs. C

N.ヴィルトの手によって，Pascalが誕生してから約20年が経過した．プログラムおよびデータの構造を明確に表現できるプログラミング言語として，ALGOL-60を基盤に開発されたPascalは，その後，Modula-2，Adaなどさまざまな ALGOL系言語の登場にも関わらず，多くの機種で利用され，教育用のプログラミング言語として高い評価を受けている．ここでは，パソコン用処理系として人気の高い Turbo Pascalを使って，Pascal 言語の魅力をさぐってみたい．

Pascalの歴史

Pascalは，1970年前後にスイス連邦工科大学にいたN.ヴィルトらによって設計された言語です．同時期にアメリカのベル研究所ではUNIXの初期のバージョンが稼動しており，C言語は1972年ころにUNIX上でのシステムプログラミング用に設計されています．その頃にはPascalは教育分野などで使われはじめていました．

もう少し前の時代からの流れを追ってみましょう．Pascalがもっとも影響を受けた言語は，ヨーロッパの数学者を中心に，数値計算のアルゴリズム記述用に設計されたALGOLという言語です．

ALGOLは，強い型と構造化構文や再帰呼び出しを取り入れて明確に定義された最初の言語というべきもので，1960年ころに成立しています．構造化プログラミングが話題になるよりずっと前の時代です．そのALGOL 60と呼ばれた規格を改良しようという作業が1960年代後半に行われていました．それ以前からコンパイラ設計の研究をしていたヴィルトはそれに参加し，C.A.R.ホアーと共に1つの原案を設計して提出しました．この原案が後にALGOL Wと呼ばれることになります．ALGOL Wは比較的シンプルにALGOL 60を拡張したものであり，残念ながら新しいALGOL 68としては採用されませんでした．しかしながら，ALGOL Wは，その構文的な美しさ，単純さから，大学でのプログラミング教育に使われていました．

その後，ヴィルトはさらに不要な機能は徹底的に省くという考え方で，Pascalを設計します．そうして，ユーザーが多様な型を定義でき，プログラムの誤りをできるだけ早い段階に検出することが可能な，優れた言語が誕生したのです．

実際の処理系は，最初はCDC6000シリーズのコンピュータ用に製作されました．その後，教育用に普及していくと種々のコンピュータ上で利用可能なポータブルな処理系の要求が強くなり，Pコードという中間言語にコンパイルするPascal Pと呼ばれるコンパイラが開発されました．このコンパイラもPascalで書かれていましたから，Pascalをあるコンピュータで利用したい場合は，Pコードを解釈実行する簡単なインタプリタを作成するだけで，移植することが可能になるわけです．こうしてPascalは世界中に広まりました．

さらに，カリフォルニア大学サンディエゴ校(UCSD)のケン・ボールズ教授らが，Pascal Pを用いて大々的にコンピュータ教育を行おうと，Pascal Pコンパイラに改良を加える作業を行っているうちに，8bitのパーソナルコンピュータ上に移植することができました．これが1979年ころにUCSD Pascalとしてソフテック・マイクロシステム社から販売され，初期のパーソナルコンピュータのユーザーの間で，最初のPascalのブームが起きたのです．同社からはその後，Pコードを機械語としてそのまま解釈実行するコンピュータがPascalマイクロエンジンという名前で発表されたりしていました．

この後，フィリップ・カーンがTurbo Pascalを作り上げ，ボーランド社を設立してこれを販売し(1983年)，爆発的に普及して今日にいたる(といってもずいぶん前ですね)のはいうまでもないでしょう．

PascalとCの比較

はじめに

ここまででおわかりのように，Pascalも長い歴史を持っています(この世界で20年といったら十分長いですよね)．その間に，Pascalの言語仕様も変化してきています．ヴィルトらのオリジナルのPascalの言語仕様を記述したものとしては，“Kathleen Jensen and Niklaus Writh, Pascal USER MANUAL AND REPORT”があります．ちょうどC言語の“Kernighan and Ritchie, The C Programing Language”に相当するものです．しかしながら，この仕様では，分割コンパイルができないとか，入出力に対話的な利用を考慮していないとか，あるいは定数として数式を用いることができないなど，大きいプログラムを書いたり，パーソナルな環境では利用しにくかったり，そのほか若干の機能不足があります．逆に見れば，ヴィルトはそれくらいぎりぎりの単純化を狙ったのですが，やはりこれでは使いにくいということで，Pascalのオリジナルの設計方針を壊すことのないように，機能が付け加えられてきまし

た．その際に標準化が行われなかったため，具体的な処理系ごとに少しずつ違った拡張が行われています．

そこで本稿では，日本のボーランド社が販売しているTurbo Pascalを念頭において記述します．

また，これからC言語との比較をしながらPascalの解説を試みるのですが，C言語の方も初期のUNIX上の処理系からずいぶんと拡張されています．とくにANSI規格になってからは，プログラムの信頼性を保証できる言語仕様にしようと，Pascal系列の言語の影響を強く受けて，かなり似てきてしまいました．本稿では，そのANSI規格で取り入れられた新しいスタイルを基準に比較していきます．また，具体的な処理系を想定する場合は，Turbo CやMS-Cを用います．

図1-1 C言語のプログラム構造

```
#include        <stdio.h>                 /* ヘッダファイルの取り込み */

#define BUFSIZ  80                        /* 定数やマクロの定義 */
char buf[BUFSIZ];                         /* グローバル変数の定義 */

int func1(int x, char *s) {               /* 関数定義 */
}
int main(int argv, char *argc[]) {        /* メインルーチン */
}
```

プログラムの構造と分割コンパイル

C言語では普通は図1-1のような書き方をします．

たいていの場合はこのような順序で要素を並べます．が，別に順序は自由です．ただし，参照する前に定義されているのが原則です(関数などは定義される前に参照すると，整数型と暗黙の定義がされてしまいますが，よいスタイルではありませんし，バグの原因にもなりやすいですね)．また，main()という関数がメインルーチンという規則があります．つまり，プログラムをコンパイルして実行すると，最初にmain()という関数が呼び出されることになっています．ソースファイルを分割しておき，コンパイルを別々に行って最後にリンクする場合でも，main()は1つのファイルだけに存在する必

COLUMN

BASIC

1964年米国ダートマス大学のジョン・G・ケメニー(John G.Kemeny)とトーマス・E・カーツ(Thomas E.Kurtz)の手によって汎用機のTSSシステム用として，ひと握りのプログラマだけでなく学生たちにも使えるコンピュータ言語，BASIC (Beginner's All-purpose Symbolic Instruction Code)が設計された．

コンピュータの素人にも使いやすく，またユーザーとコンピュータのコミュニケーションを助けるような処理系を目指し，従来の処理系のように厳密な型チェックや厳格な構文を排し，初心者が試行錯誤しながらマスターできる対話型言語としてBASICは設計された．

BASICが産声をあげた10年後の1974年に，BASICを爆発的に普及させる事件が起きた．MITS (Micro Instrumentation and Telemetry System) 社の発売した「Altair」の登場である．メインメモリがわずか256バイトしかなかったこのコンピュータ上で動くBASICが，翌年ビル・ゲイツ (William H. Gates)とポール・アレン (Paul G. Alen) の手によって開発され，このBASICが，自分のコンピュータを持って，思う存分プログラムしたいというハッカーに歓迎されたのである．ただしケメニーとカーツの開発したBASICと，ゲイツとアレンのそれは，大きく異なっていた．一般にBASICの欠点とされるマルチステートメントやコマンドの短縮，インタプリタ型などは，すべてパソコンのメモリの制約からきたものであり，ケメニーとカーツのBASICには存在しないものだった．以後メモリ制約の多いパソコン用言語として，この特徴を持ったBASICが採用されていったのである．

BASICといえば，GOTO文の多用からスパゲッティのようにもつれたプログラムになってしまい，書いた本人さえ理解できない言語とされていた．しかし現在では，構造化プログラミングの手法を取り入れ，モジュール化を容易にするために行番号を排したBASICが主流になっている．

図1-2 Pascalのプログラム構造

```
program sampleProg;                    { プログラム名の定義 }
uses                                   { 使用するユニットの定義 }
          Graph;
label                                  { ラベルの定義 }
          9999;
const                                  { 定数の定義 }
          bufsize = 100;
type                                   { ユーザー定義型の定義 }
          Week = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
var                                    { グローバル変数の定義 }
          aDay: Week;

function Func1(I, J: Integer): Real;        { 関数定義 }
begin
          .
          .
end;

procedure Proc1(X: Integer);                { 手続きの定義 }
begin
          .
          .
end;

begin                                       { メインプログラムの定義 }
          .
          .
end.
```

要があります．分割コンパイルを行う場合に，どのファイルをリンクするかは，べつにmakefileなどで指定する必要があります．少なくとも，プログラムの字面からはどのファイルとリンクすべきかはわかりません．あくまで，C言語の処理系はオブジェクトファイルを作成するまでしか面倒をみないのです．

対して，Pascalのプログラムは図1-2のような構造をしています．

必ず先頭にプログラムの名前を書き，プログラムの最後はピリオドで終わるという書き方になっています．元のPascalではラベルの定義から変数定義までは，この順に多くとも1回だけ現れることになっています(つまり不要なら省略可です)．Turbo Pascalでは，この制限は若干ゆるめられています．すなわち，順序が入れ代わっても，2か所以上の場所に分かれてもよいことになっています．ただし，参照される前に定義されるという原則は厳格に守られています．一切の省略時解釈はありません．また，制限がゆるめられているといっても，とくに理由がない限りこの順序を守ったほうが，プログラムが読みやすいと思います．

図1-2では，プログラム名の定義の次の行にusesという予約語ではじまる，使用するユニットの定義が行われていますが，これがTurbo Pascalの分割コンパイルのための構文です．このユニットの概念というのはAdaにおけるPackageの概念などに強く影響されたものです(Modula-2と類似のものです)．簡単に説明しますと，ユニットというものは，インターフェース部と実現部からなります．実現部というのがC言語でいう普通に分割されたソースファイルに相当する部分です．インターフェース部というのはC言語ではヘッダーファイルとして記述される，別ファイルのソースファイルから公開されている関数や変数の定義が納められている部分です．Turbo Pascalではこの2つの部分を同じファイルに記述しておきます．またユニットをコンパイルして出力されるオブジェクトファイルの内部にも，特別な形式

図2 Pascalの関数定義と手続き定義

```
関数定義
    function func1(foo, bar: integer): integer;     { 関数名と引数，型の定義 }
    label                                  { ラベルの定義 }
          ...;
    const                                  { 定数の定義 }
          ...;
    type                                   { ユーザー定義型の定義 }
          ...;
    var                                    { ローカル変数の定義 }
          ...;

    function ...;                          { 関数定義 }
    begin
          .
    end;

    procedure ...;                         { 手続きの定義 }
    begin
          .
    end;

    begin                                  { 関数本体の定義 }
    func1 := returnValue;                  { 戻り値の決定 }
    end;
手続き定義
    procedure proc1(foo, bar: integer);    { 手続き名と引数，型の定義 }
    label                                  { ラベルの定義 }
          ...;
    const                                  { 定数の定義 }
          ...;
    type                                   { ユーザー定義型の定義 }
          ...;
    var                                    { ローカル変数の定義 }
          ...;

    function ...;                          { 関数定義 }
    begin
          .
    end;

    procedure ...;                         { 手続きの定義 }
    begin
          .
    end;

    begin                                  { 関数本体の定義 }
          .
    end;
                  '...'は省略の記号です
```

C リスト1

```
(1)
    void swap1(int x, int y) {
    int tmp;
        tmp = y;
        y = x;
        x = tmp;
    }
(2)
    void swap2(int *x, int *y) {
    int tmp;
        tmp = *y;
        *y = *x;
        *x = tmp;
    }
```

リスト1 Pascal

```
(3)
    procedure swap1(x, y: integer);
    var tmp: integer;
    begin
        tmp := y;
        y := x;
        x := tmp
    end;
(4)
    procedure swap2(var x, y: integer);
    var tmp: integer;
    begin
        tmp := y;
        y := x;
        x := tmp
    end;
```

でインターフェース部の情報も格納されています.

そうして，あるプログラムをコンパイルするときに，あるユニットが使用されていることがわかると，自動的にそのユニットのインターフェース部を取り込んで，公開されている変数や関数，手続類の定義を利用できるようにします．そして，同時にどのファイルをリンクすべきかも知ることができるわけです．もちろん，あるユニットから別のユニットを利用することもできます．

このようにしてTurbo Pascalでは，分割コンパイルを行う場合でも，厳密な型チェックを行ってエラーの検出を行えるようにしています．

また，コメントを書くときは，C言語では /* と */ のペアで挟むことになっていますが，Pascalでは { と } か (* と *) のペアで挟むことになっています．

説明するのが遅れてしまいましたが，Pascalでは，求めた値を返すルーチンを関数，値を返さないものを手続きといって区別します．関数は式の中で使うことはできますが，手続きのように呼び出すことはできません．手続きは逆です．

C言語では，形式的には両方とも関数ということになっています(中には値を返さない変な関数もあります)．また，値を返す関数は，式の中で使うこともできますし，式自体が文としての役割を持っているので，手続きのように使うこともできます．

Pascalでは，関数定義と手続きの定義は図2のようになっています．最初の1行を除けば，ほとんどプログラムの定義と相似になっていることに気がつきます．C言語と比べて大きな特徴は，ローカルな関数や手続きを持つことができることです．このローカルな関数や手続きは，ローカル変数と同じ

図3

```
program sample1;

    function func1:integer;

    end;

    function foo:integer;
        function func2:integer;

        end;
    begin
                                ..........(a)
    end;

    function bar:integer;
        function func3:integer;

        end;
    begin
                                ..........(b)
    end;

end.
```

ようなスコープを持ちます．図3を見てください．aの位置からでは，func1とfunc2を呼び出すことができます．bの位置からはfunc1とfunc3とfooが使えます．たとえ，func1とfunc2が同じ名前だとしても，2重定義したことにはなりません．また，func1とfunc2が同じ名前だったとすると，aの位置からはfunc2だけが呼び出せます．

ローカルな関数や手続きをうまく利用すると，名前の衝突を回避することが可能です．関数の戻り値は，C言語ではreturn文を利用しますが，Pascalでは関数名に対する代入によって値を決めます．

手続きや関数の仮引数の定義部分にvarを付けるか付けないかで，値渡しかアドレス渡しかの区別が行われます．リスト1を見てください．

これらの手続きが2つの変数の内容を入れ替える手続きを定義しようとしたものならば，(1)と(3)は同じ種類の間違いをしています．(2)と(4)が正解です．C言語もPascalも，普通は関数や手続きの引数は，値を求めてコピーして利用しているために，(1)や(3)ではコピーの間の入れ替えをしているだけでもとの引数の内容を操作していません．そこで，C言語では明示的にポインタを渡すように宣言します．Pascalでは，varを仮引数に付けておけば，実引数の変数のアドレスが渡されて，直接に実引数を操作するような動作が行われます．

型の定義

CもPascalも強い型をもった言語です．変数や値はさまざまな型を持ちます．異なった型の間の演算や代入は，ある組み合わせでは自動変換され，ある場合にはエラーになります．どちらかというと，C言語ではできるだけ型の組み合わせが自由にできるように，意味のある演算をしようと解釈します．たとえば，char型とint型はint型で演算するようにします．しかし，これはPascalでは許されません（ただし，C言語のchar型は，Turbo Pascalでいうshortint型の代わりに使われることも多いのですが，この場合，shortint型とint型の演算は許されています）．たとえば，C言語では'A'+10は可能ですが，Pascalではエラーになります．

まず，Pascalでいうスカラー型ないし単純型から比較してみましょう．Pascalでは順序型と実数型が許されています．順序型とは，簡単には整数のようなもので，離散的で，その型に属する値の間に順序関係があるものをいいます．

Turbo Pascalでは，5種類の整数型，論理型，文字型が定義されており，さらにユーザー定義の列挙型と部分範囲型があります．

これらをC言語と対応させてみましょう（表1）．

列挙型は，括弧()の中に識別子を並べた形になっています．C言語と比べてみましょう（図4）．

順序型の性質としてはある値の次に大きい値を求めるSucc関数と，逆に1つ前の値を求めるPred関数が使えること，順序型の値から普通の整数に変換するOrd関数が使えることがありますし，大小関係も決定できます．

表1　PascalとCのデータ型

Pascal	C	値の範囲（Pascalにおける表記法）
shortint	char	-128..127
integer	int	-32768..32767
longint	long	-2147483648..2147483647
byte	unsigned char	0..255
word	unsined	0..65535
---	unsined long	
boolean（論理型）	(int で代用)	True，False
char	(char で代用)	
列挙型	enum	
部分範囲型	---	

図4　PascalとCの列挙型

```
Pascalの列挙型
        Week = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
Cの列挙型
        enum Week {Sun, Mon, Tue, Wed, Thu, Fri, Sat};
```

図5　部分範囲型

```
Month = 1..12;
WeekDay = Mon..Fri;
```


PC-9801 I/Oポートマップ
機種：①②③　ポートアドレス：XXXX00XX0011XXA0　RS-232Cインターフェース　使用チップ：μPD8251A（8251相当）

表 2　構造をもつ型

Pascal	C
配列型	配列
集合型	---
レコード型	構造体，共用体
ファイル型	(FILE *に相当)

図４のような場合, C言語でも Pascal でも Mon < Wed は True になります．C 言語では(Sun + 1)の値は Mon と等しくなりますが，Pascal では(Sun + 1)という式は不正な型によるエラーになります.同じことをするには, Succ(Sun)という式を書きます.

さらにC言語には対応するもののない型として，部分範囲型があります．これはすでに定義された型の一部分だけを利用するための型です．部分範囲型を指定するには,最小値と最大値の間に'..'を入れることで行います(図５).

もちろん，Month型の変数に30のような範囲を越えた値を代入することはエラーになります．また，後述しますが，配列型の添字範囲を指定するのにも用います.

実数型は，C 言語では double と float が基本，Pascal では Real が基本ですが，計算機に使われる種々の精度によっていくつかの型が定義されています.

また，文字列を扱うための文字列型も用意されています.

さらに，C言語も Pascal も構造をもった型を定義できます．これも対応関係を表にしてみましょう(表２). Pascal の配列型はC言語と異なり，添字の下限と上限の両方を指定することができます．もしも－10から10までの添字が使えるAという整数型の要素からなる配列型が必要ならば,

```
array [-10..10] of integer;
```

で，すみます．もちろん，多次元配列も可能です.

また，レコード型はCの struct と union の両方の機能を包含しています.

ファイル型は入出力用の特別な型です.

定数と変数

C言語では定数はマクロの一種で，本来はプリプロセッサでコンパイラが処理する前にソースプログラムの該当する識別子を書き換えてしまう方法で実現しています．もっとも最近のコンパイラでは，コンパイル時間を節約するためにコンパイラの中にプリプロセッサ相当の部分を組み込んでいるものが多くなっています.

Pascal では言語仕様の中に，しっかりと定数が規定されています．定数と変数との違いは基本的には，定数は const に続く行で定義されるのに対して，変数定義は var という予約語がはじまりを表すということ，定数には初期値が与えられておりプログラムのほかの部分で値を変更することができないことです.

Pascal の変数定義とC言語のそれを比較すると，C言語では型が先に来るのに Pascal では':'で区切って後ろに位置することが目新しいくらいでしょう．そんなに差は感じられません.

演　算

Pascal は, 四則演算プラスアルファの演算子を持ちます．ただし，C言語に慣れていると奇異に感じることがいくつかあります.

表３を見てください．記号ではなく単語で表されている演算子があります. /は商を求めますが，結果は実数型になります. div も商を求めますが, 小数点以下はゼロに向かって切り捨てた整数型になります. shl と shr はシフト演算子です．それぞれC言語の<<と>>に対応しています.

とくにC言語に慣れた人の陥りやすい誤りとして，論理演算の優先順位が異なっていることが挙げられます．たとえば,

```
a < b and b < c
```

という式は，C言語に慣れていると,

```
(a < b) and (b < c)
```

のように見えますが, Pascal では and 演算子の方が優先順位が高いので,

```
a < (b and b) < c
```

と解釈されて，型検査に引っ掛かってエラーになるか，運が悪いとエラーも出ないかわりに望む結果も得られないはめになります.

Pascal では，代入は演算子ではなく，代入文として式とは別に存在します．が，表記法の差についてだけ，ここでふれておきます．Cでは等しいという比較を表す演算子を＝＝，代入を＝で表します．Pascal では，等しい比較演算を＝で，代入を：＝で表します．また，等しくないというのも，Cでは！＝, Pascal では<>と，ちょっと

表3　Pascalの演算子（上ほど優先順位が高い）

not，+，-　（以上は単項演算子）
*，/，div，mod，and ，shl，shr
+，-，or，xor
<，<=，>，>=，=，<>，in

だけ違いがあります．

さらに，関数呼び出しについて，とくに引数のない関数にはとまどうことがあります．C言語では，関数呼び出しには必ず関数名の後ろに()が付きます．ところが，Pascalでは，引数がなければ，()は付けません．ですから，C言語のプログラムを読んだ後に，Pascalのプログラムを読むと，つい，引数のない関数を変数と思い込んで変数定義を探し回ったりしてしまいがちです．逆にプログラムを書く場合の心構えとしては，変数には変数らしい名前を，関数には関数であることがすぐにわかる名前を付けるようにするべきです．

これは余談ですが，C言語では演算子の絡み合った読みにくい式を書くことができます（もちろん，そうならないように書くべきですが）．Pascalでは比較的に単純な式が並ぶことになるはずですが，冗談で，引数を持たないIとかJとかValのような名前の関数を定義しておき，CheckFlagとかCompareItとかのグローバル変数を定義しておくと，とてつもなくわかりにくいプログラムを書くことがPascalでも可能になります．

制御構造

C言語にもPascalにも，ほぼ同様の制御構造があります．Pascalでいうif文，case文，while文，repeat文，for文のそれぞれに対応するものがC言語にもあります（図6）．if文やwhile文についてはほとんど同じといってもよいでしょう．case文とfor文については少しコメントをしておきます．

Pascalのcase文では，分岐先を表

COLUMN

FORTRAN

1954年，コンピュータ市場の制覇を目指していたIBM社はジョン・バッカス（John Backus）をリーダーとして，新型の704コンピュータを対象とした自動プログラミング・プロジェクトを発足した．このプロジェクトによって生まれたのがFORTRANである．当時この言語は“IBM Mathematical Formula Translating System”（IBM数式変換システム）と呼ばれ，対象ユーザーを明確に科学者とエンジニアに設定し，ユーザーのニーズに合わせた言語設計がされた．まだ対話型プログラミング言語が存在していない時代のことである．

1966年にはANSIで規格化（FORTRAN66）され，いよいよ普及することになる．これは当時FORTRAN IVと呼ばれていた言語仕様をまとめたものだ．FORTRAN IVは既存のサブルーチンをプログラムに組み込むための手続きを用意しているため，新しいプログラムを作るとき省力化が可能となる．うまくすれば，ライブラリのサブルーチンを組み合わせるだけで，新しいプログラムが作成できることになる．ただしFORTRAN66には多くの制限もあった．そこで登場したのが，FORTRAN66の制限を取り除き，多くの技術者の要望を取り入れたFORTRAN77である．さらに91年4月にはFORTRAN90の仕様がまとめられた．おもな変更点は，7文字以上の識別子やDO-WHILEループなど，各処理系が独自に拡張していた部分の追認，モジュール概念やポインタの導入にともなう記憶構造の変更などである．

先日発売されたMicrosoft FORTRAN Version 5.1は，このFORTRAN90規格を含み，QuickWinというライブラリをリンクすることでMS-Windows 3.0上で動作するアプリケーションを開発できるなどの特徴をもっている．

最近はCやパスカルのライブラリも爆発的に増えているが，FORTRANの長年培われてきた膨大なライブラリに比べればまだまだ少ないといえるだろう．とくに科学技術計算の分野においては，ありとあらゆるFORTRANのライブラリが揃っている．このライブラリの差こそが今日のFORTRANを支えているのである．

PC-9801 I/O ポートマップ
機種：―　ポートアドレス：XXXX0IXX00IIIAAI　システム予約

図 6
制御構文

```
Pascalのif文                              Cのif文
if 条件 then 文1 else 文2                 if(条件) then 文1 else 文2

Pascalのcase文                            Cのswitch文
case 式 of                                switch(式) {
        定数1:       文1;                              case 定数式1:
        定数2:       文2                                    文1
        else     文n
end
                                                       case 定数式2:
                                                case 定数式3:
                                                       文2
                                                default:
                                                            文n
                                          }

Pascalのwhile文                           Cのwhile文
while 式 do 文                            while(式) 文

Pascalのrepeat文                          Cのdo while文
repeat 文 until 式                        do 文 while 式


Pascalのfor文                             Cのfor文
for 変数 := 式1 to 式2 do 文              for(式1; 式2; 式3) 文
for 変数 := 式1 downto 式2 do 文
```

リスト 2
Pascal

```
case x of
     1, 3, 5:      y := y - x;
     100..200:     y := x - y;
else               y := x
end
```

す定数の部分に複数の定数を‘,’で区切って並べたり，部分範囲型の定数を置けます．すなわち，リスト2のような書き方ができます．

また，Cではbreak文を配置しなければ,次のcaseラベルのついた部分も続けて実行してしまいますが，Pascalではリスト2の例のような場合，xが1だとすると，y：＝y－xを実行するだけでcase文から抜け出ることになっています．技巧的なプログラムは書けないかもしれませんが，単純明快な多方向分岐を行えるようになっているのです．

Cのfor文は，最初に式1を実行して，式2が成立している間は文を実行した後に式3を計算することを繰り返すという形式になっています．それぞれの要素は，必要なければ省略してもかまいません．Cの代入を含む多様な式の表現を用いると，for文本来の，変数をある値から別の値まで変化させながら文を実行するという目的以外に多種類のループを実現可能な構文に拡張されています．

対してPascalのfor文は，あくまで本来の形式に固執しているように見えます．制御変数の型は順序型である必要があり，実数型などは許されません．また変数の増分も，1（toの場合）か－1(downto)に制限されています．しかし，この制限も信頼性の高い記述のためには必要なものです．たとえばBASICのように制御変数に実数型を許し，増分も自由にすると，ループ回数に±1回の誤差が入る可能性がでてきます(手元の計算機で，0.01を10000回加えてから100を引いてみてください．たぶん，答えは0にならないと思います)．

C言語のfor文などは，利用する人のスタイルによってはずいぶんと書き方が異なる場合があります．それに対して,Pascalでは誰が書いても同じようなスタイルになるように制限を設けているともいえます．

入出力と標準ライブラリ

C言語では，入出力の機能は言語の文法には規定されていません．ライブラリ・ルーチンによって規定されることになっています．もちろん，これでは困りますので，昔から標準的なライブラリの仕様がありましたし，ANSI規格ではライブラリで用意する入出力関数群についても規定されています．そして，低レベルのファイルの読み書きから高レベルの書式付き入出力関数まで，非常に多くの，豊富な機能をもった関数が用意されています．

Pascalでは，本来，入出力の機能は言語処理系の組み込み関数や手続きとして規定されていました．通常の手続きとは異なり，不定個の引数を取る手続きが必要になったりするからということもありますし，教育用の小規模のプログラムを作るために設計されたので，最小限の入出力をサポートするだ

けに留めたためかもしれません．それになにより分割コンパイルが不可能だったので，ほかから供給されるライブラリをリンクすることができないという理由があったのだと思われます．しかし，現在のTurbo Pascalではユニットによる分割コンパイルないしモジュール化プログラミングが可能です．そして，コンピュータのテキスト画面やグラフィック画面に対する操作，システムコールを含むさまざまなユニットが最初から付属しているので，Cと比べても遜色はないと思われます．

C言語におけるファイルは，基本的に文字の並びとして単純化されています．Pascalではテキスト型のファイルのほか，型を持ったファイルを定義できますから，その分だけ使いやすい面もあります．たとえば，あるレコード型のファイルがあったとすると，そのレコード単位でのランダムなアクセスが可能になります．1回の入出力で，レコード単位で読み書きができます．データーベース的なプログラムなどが書きやすい仕様です．

まとめ――これからプログラミングを学ぼうとする人へ

以上のように，PascalではC言語と異なり，ある種の作法を強制します．文字型と整数型の間の演算は明示的に型変換を行う必要があるなど，C言語に慣れている方にはうっとうしい性質もあります．しかしながら，C言語ではちょっと文字列を操作しようとする場合には，明示的にポインタを取り扱う必要があります(配列操作のように見せかけるシンタックス・シュガーもありますけど)．また低水準の操作も気楽にできます．しかし，Pascalでは動的な記憶割り当てなどを利用しなければ，明示的なポインタ操作などはありません．

全体を見渡せば，C言語はプログラムをバリバリ書くハッカーのための言語という特徴があるでしょう．扱うのはあくまでハッカーなわけですから，きっとプログラムに変なバグなど放りこまないでしょうし，たとえ放りこんだとしてもすぐ見つけられるのでしょう．そのぶん，ハッカーの足手まといをしないように，言語としては柔軟性に富んだ仕様ですし，作業のリズムを狂わせないようにタイプする手間を減らす工夫もされています．

しかし，これからプログラムを学ぼうとする人にとっては，記述の自由さから，プログラミングをする上での標準となる書き方を見失いがちですし，何よりも初期化されていないポインタ，あるいはとんでもないところを指しているポインタに起因するわけのわからないバグを発生しがちです．自動的な型変換も，C言語では関数の引数などで無頓着なことをしていると，ヘッダファイルのインクルードを忘れたりすると，とてつもない目にあうこともあります(たいていはコンパイラの出す警告で気がつきますが)．

たとえ2倍の文字をタイプしてもよいではありませんか．減らすべきなのは，デバッグにかける時間である．というのがPascalの精神です．などと，よくいわれていますね．

ほんとのところは，C言語でもきっちりとしたプログラムは十分に書けますし，そのおかげで多くのプログラムがC言語で作成されて世の中に出てきています．しかし，そういった達人たちの手になるプログラムを読めば，それぞれの人がきっちりとした書き方のスタイルを持っていて，C言語の仕様がいかに自由だとはいっても，その自由さに型をはめて利用していることがわかるはずです．

また，C言語はポインタなどが丸見えだといっても，アセンブラでしっかりとしたプログラムを書いてきた人にとっては，変に抽象化された言語よりは中身がはっきりつかめるので，Pascalなどよりかえって学びやすいという側面があります．また逆に，そのようにコンピュータの箱の中で何が行われているかをはっきりと意識している人のために設計されているのです．

しかし，これからなにかプログラミングを学ぼうとする人にとっては，最初から重荷を背負うことになります．車の運転を学ぼうとするときを考えてください．車が，つねにエンジンのピストンやカムの動作に気を配らなければ動かすことのできないようなものだとしたら，運転のしかたを学ぶのは困難なことでしょう．もちろん，まったく知らなくてもよいドライバーになれるかというとそれも疑問です．とはいえ，最初はとにかく運転作業の初歩を学び，とりあえず動かせるようになれば，かえってエンジンの様子なども学びやすくなります．そうして，少しずつ熟達していけば，エンジンの異常にもすばやく感じ取ることのできる優れたドライバーになることができるでしょう．

プログラミングを学ぶのも同じことです．最初は単純なモデルから出発したほうがよろしい．また，最初のころは，慣れていると無意識のうちに回避してしまうような過ちをしがちです．ですから，比較的単純なPascalからはじめましょう（とはいえTurbo

Pascalは十分すぎるほど拡張されて複雑になっています．でも最初はそのような部分を無視することができます)．うっかりミスもしつこいぐらいにコンパイラがはじき出してくれます．

そうして，十分に知識がついたら，必要に応じてC言語でもBASICでも使えばよいのです．専門家の間では手続き型と呼ばれている種類の，皆さんが触れることのできる言語のほとんどは，Pascalと見かけほどは差がないものですから，すぐに乗り換えられます．

プログラミング言語の比較紹介などした後では書きづらいのですが，とくに注意しておきたいことがあります．プログラミングを学ぶことにおいて，言語の習得というのは，実は枝葉末節に属する事柄です．本当に学ばなければならないのは，問題をコンピュータに解かせるにはなにをすればよいか，です．1つの問題を，解きやすい部分問題に分解して，問題に応じて適切なデータ構造を設計し，プログラムを構成していく方法です．

これを自分のものにすれば，プログラミング言語として何を利用するかなど，たいした問題ではありません．初心者の方で，プログラムがうまく書けない，挫折ばかりしているという目にあいがちなのは，こちらの勉強が足りないことが多々あります．

この分野でしばしば紹介される良著として，Pascalの設計者であるヴィルトが書いた「アルゴリズム＋データ構造＝プログラム」がまずあげられるでしょう．また最近は探せばさまざまなデータ構造やプログラムの設計に関する良い本が出版されています．たいていの場合，そういう本の中でプログラムの具体例を示すのに使われるのは，PascalかC言語になっていますので，ちょうどPascalの勉強と並行して読み進むことができると思います．もちろん，「アルゴリズム……」もPascalを使っています．

最後に，これからPascalを学ぼうとする人にとっての，すぐれた入門書も紹介しておきましょう．Turbo Pascalのマニュアルです．ユーザーズ・ガイドの前半と，必要に応じてプログラマーズ・ガイドも参照すればよいでしょう．なんと，この入門書は，購入するともれなくPascalコンパイラが付いてきます．値段もおそらく1万円台で購入できるでしょうから，結構なページ数であることを考えれば，変な入門書を何冊か購入するより付属のコンパイラ分だけ得だと思います．

では，Pascalで豊かなプログラミングの世界に旅立ってください．

COLUMN

COBOL

1960年4月，データシステム言語委員会(CODASYL)の臨時委員会はCOBOL(Common Business Oriented Language)の仕様の初版COBOL-60を発行した．事務処理アプリケーションの開発に適する処理系COBOLの登場である．このときのCOBOLには「ある特定のメーカーが不当な優位に立たない」といった配慮がされていた．これが，多くのコンピュータの上でCOBOLがビジネスアプリケーション開発用言語としての地位を保つことができた要因の1つである．COBOLの言語体系は英語の構文に近く，英語の動詞で始まり，ピリオドで終る．またアルゴリズムよりもデータに重点がおかれている．こうした読み易さはプログラマだけでなく，ユーザーであるビジネスマンでもプログラムが理解できることを示している．

COBOLでは機種ごとに依存する環境部が独立しているため，COBOLのプログラムは環境部さえ交換すれば機種に関係なく実行することが可能となる．ただし，COBOL-60の仕様にはソート機能とレポート作成機能といった肝心な機能が設定されていなかった．

この後，COBOL-61，65，74などを経て，COBOL-85と改良されていった．COBOL-85による仕様追加は，コンピュータの主記憶容量の増加にともなった取得可能な配列のサイズの大型化，ループ命令をほかの言語と統一したことなどである．ただし，あいかわらず再帰呼び出しが使えず，また今後も使えるようにはならないだろう．次のCOBOLで予定されているおもな機能はオブジェクト指向である．

パソコン用COBOLは，MicrofocusやWatcom社など数社から発売されている．ただし，実質的にはMicrofocus COBOL/2の独占状態にある．Microfocus COBOL/2は米国ではWindows版，OS/2 PM版，DOSエクステンダ版などがあるが，いずれも日本語化はされていない．Watcom COBOLは日本ではライフボートから発売されているが，教育市場を狙った製品位置付けで，大規模なアプリケーションを組むことは難しいといわれている．

4GL(第4世代言語)の高波にさらされながら，COBOLも時代の要求に合わせて進化しているのである．

特集
Cじゃなくてもいいじゃない

株式会社NJK
鈴木朋夫

チームプログラミングに最適の言語仕様

MODULA-2 VS. C

ヨーロッパではPascalと並んで教育用言語として人気の高いMODULA-2は，Pascalのもつ構造化の概念をさらに押し進めて，徹底したモジュール化を意識した言語であり，多くのPascal処理系がPascal本来の言語仕様を独自に拡張する形ですすめてきた，分割コンパイルなどの機能をその仕様に取り込んでいる．MODULA-2という言語のもつ特徴をC言語との比較によって，明らかにするとともに，アプリケーション開発におけるMODULA-2の採用例なども紹介する．

MODULA-2は「アルゴリズムとデータ構造」の著者でありPascalの設計者でもあるニクラウス・ヴィルト教授により1978年に定義されました．この言語は，言語の設計からワークステーションを構築するLilithプロジェクトの一環として，OSからアプリケーションまで単一の言語で記述できることを目標に設計されました．このプロジェクトにより教授は1984年度のACMチューリング賞を受賞されました．

NJKでは6年ほど前から米国ロジテック社製のLOGITECH MODULA-2を国内マシンに移植し，販売を続けています．ここではMODULA-2の特徴をとくにチームによるプログラミング開発という視点からCと比較してご紹介します．

チームプログラミングとプログラミング言語

プログラミングは，個人的な色彩の強い作業です．ですから普通プログラミング言語は，プログラマ1人1人との関係から解説されます．しかし今日のプログラマ（とくに職業プログラマ）は1人で仕事をすることはまれで，ほとんどのシステム開発ではチームプログラミングを行います．そうした開発作業は「分析→設計→製造→保守」といった工程を繰り返しますが，実際にはじめてみると次のような制約や問題につきあたります．

1.納期の制約（提出期限の制約）

2.予算の制約

3.コミュニケーションの問題

図1
プログラミング言語に要求される特質

1．部品になる ← 既存のパーツを転用出来る．
2．部品の組み合わせの正当性を検出する ← 他のプログラマとの間で，各自のパーツのインターフェースについて間違いなく連絡する．
3．文書になる ← プログラムが読み易い．
4．属人性が低い（同じになる） ← 誰の書いたプログラムでも読み易い．
5．文法チェックが厳格 ← 間違いに関するチェックが厳しい．
6．習得性が高い ← 簡単に覚えられる．
7．安全性が高い ← 安全に使える．

自分の楽しみだけにプログラミングできる純粋なアマチュアプログラマを除けば，多くのプログラマは，プロジェクトの納品日，懸賞や卒業制作プログラムの提出期限といった納期の制約から逃れられません．加えて職業プログラマには予算の制約があります．

こうした制約は高い生産性を要求します．プログラミングが好きなプログラマは個人個人で技量を上げる努力を惜しみませんが，チームを組んで開発するプログラマには，さらに仕様変更，チーム内のプログラマの技量の差から製造工程で起きる情報伝達の相違，チーム内のプログラマの入れ替わり，開発情報の散逸で保守工程に生じる情報消失といったコミュニケーションの問題を解決しなければなりません．

システム開発の各工程で私たちはさまざまなツールを使いますが，設計の最終段階と製造，保守段階でシステム開発に採用するプログラミング言語は，直接プログラム実装の効率や成否に大きなウェイトを占めています．とくに保守段階でのシステムのソースは，ある期間面倒をみなければならない宿命であるといえます．こうした状況下で使用するプログラミング言語には，1人でプログラミングするときとは異なる特性が求められます（図1）．

チームプログラミングに望ましい言語の特性の第1は部品になることです．これによってテスト済みのコードを部品として蓄積，再利用して，テスト範囲とコーディング時間を短縮することができます．第2は組み合わせの正当性を検出することです．これによってインターフェースの変更に確実な伝達手段を用意し，コミュニケーションの誤りを防止します．第3は文書になることで，ソースの読みやすさを高めます．第4は（誰が書いても）同じになることで，さまざまな表記方法を抑制し，プログラムに表れる差異をアルゴリズム（解法）の差だけにします．

これらの特性は保守段階に有効です．最悪の場合（これが多くの場合でもあるのですが）保守者は製造者ではありませんし，ドキュメントはソースが更新されるほど更新されるとは限りません．ソースを頼りに保守作業を行うのはよくあることです．プログラムが部品化され，ソースの表記が文章に近く，構文が十分整理されていれば，保守者はすぐにアルゴリズムを検討できるのです．

文法のチェックが厳格なことは，間違いを犯すプログラマを保護する機構として用意されるべきです．また習得性が高いことは，数を問題にする製造段階の人的資源の供給と確保に必要です．しかし，いくら簡単に覚えられても，使う段階の危険性を少なくできなければよい結果は得られません．短期間で育成された経験の浅いプログラマ

によってプロジェクト全体が危険にさらされるのはなぜでしょうか．彼らの経験の少なさ，不慣れ，技量の未熟を問題にするのはアンフェアです．むしろ問題の一因は使用言語がチームプログラミングに適した特性を欠いているので，彼らの記述した危険なロジックが他へ影響することを防げないことにあるのではないでしょうか．安全性が高いことは，開発で使う言語がこれまで述べたような特性をもち，各プログラマの細部の実装の技量（注意深さ）による差を吸収し，経験の浅いプログラマのプログラムにも一定の品質を保証することで達成されるのです．

図2 モジュール

```
モジュール
  ├─プログラムモジュール
  └─ライブラリモジュール
      ├─定義モジュール（デフィニションモジュール）＝　外部仕様
      └─実現モジュール（インプリメンテーションモジュール）
```

MODULA-2 vs. C

では，具体的にMODULA-2とCのソースを比較しながら，MODULA-2のもつ特徴を見ていきます．

モジュールを示す明快な構文

システム開発に際しては，プログラムを機能分割した部品の組み合わせで構成する技法が広く採用されてます．しかし理論的に「1つの部品」として決定した機能群をソース上でそれとわかる明確な表現形式をもつ言語は限られます．

MODULA-2は言語仕様に「モジュール」を示す構文を定義しています．これはたとえばMyersの「ソフトウェアの複合／構造化設計」で示された情報的強度モジュールに明確な構文を与えたものと見ることができます．モジュールを構文にもち込んだ利点は3つ挙げられます．

C　リスト1　簡単なサンプル　C

```c
#include <stdio.h>

static int fibonacci(int);
static int fibonacci(wk)int wk;
{
    int x1,x2,x3,i;
    x1=0; x2=1;i=0; x3=x2;
    while ( i < wk ) {
        x2=x2+x1; x1=x3; x3=x2; i++;
    }
    return (x1);
}

static void Calc(int *);
static void Calc(ans)int *ans;
{
    int wk ;
    scanf("%d",&wk);
    *ans = fibonacci(wk);
}

void main(void);
void main()
{
    int i,Max;
    Calc(&Max);
    for( i=1; i<=Max; i++ )
      printf("Hellow C ");
    printf("¥n");
}
```

リスト1　簡単なサンプル　MODULA-2

MODULA-2

```
MODULE Sample;
FROM InOut IMPORT
  ReadInt,WriteString,WriteLn;

  PROCEDURE fibonacci(wk:INTEGER):INTEGER;
    VAR x1,x2,x3,i:INTEGER;
  BEGIN
    x1:=0;x2:=1;x3:=x2;i:=0;
    WHILE i<wk DO
      x2 := x2+x1;x1:=x3;x3:=x2;INC(i);
    END;
    RETURN x1
  END fibonacci;

  PROCEDURE Calc(VAR ans:INTEGER) ;
    VAR wk : INTEGER ;
  BEGIN
    ReadInt(wk);WriteLn;
    ans := fibonacci(wk);
  END Calc;

  VAR i,Max : INTEGER;


BEGIN
  Calc(Max);
  FOR i := 1 TO Max DO
   WriteString('Hellow Modula-2 ');
  END ;
  WriteLn;
END Sample.
```


PC-9801 I/Oポートマップ
機種：①②③　ポートアドレス：XXXX00XX0110AAA0　CRTコントローラ（テキスト）　使用チップ：μPD7220（7220相当）

リスト2-1
ライブラリモジュール(定義モジュール)
MODULA-2

```
(*
Title    : StdIo.DEF (sample)
LastEdit: 10.Jan.92
Author   : T.Suzki
System   : LOGITECH MODULA-2 version 3.0
*)
DEFINITION MODULE StdIo;
  TYPE
    StdIoType = (stdin,stdout,stderr,stdaux,stdprn);

  PROCEDURE Write(dev:StdIoType;ch :CHAR);
  (* 標準デヴァイスに1文字書き出す *)

END StdIo.
```

リスト2-2
ライブラリモジュール(実現モジュール)
MODULA-2

```
(*
Title    : StdIo.MOD (sample)
LastEdit: 10.Jan.92
Author   : T.Suzki
System   : LOGITECH MODULA-2 version 3.0
*)
IMPLEMENTATION MODULE StdIo ;
FROM SYSTEM IMPORT ADR,ADDRESS,DOSCALL ;

  PROCEDURE Write(dev:StdIoType;ch :CHAR);
    VAR outHandle  : CARDINAL ;
        bufadr     : ADDRESS  ;
        writebytes,
        error      : CARDINAL ;
        buf        : CHAR     ;
  BEGIN
    buf       := ch ;
    outHandle := ORD(dev);
    bufadr    := ADR(buf);
    DOSCALL(40H,outHandle,1,bufadr,writebytes,error);
  END Write;

END StdIo.
```

1.情報隠ぺいが自然に達成できる
2.手続き名の衝突を大幅に回避できる
3.ソースの完結性が明示される

MODULA-2には4種類のモジュールがあります．プログラムの開始点/終了点をもつプログラムモジュール(メイン，リスト1)，部品として組み合わされるライブラリモジュール(定義，実現モジュール)，そして各々の中に定義可能なローカルモジュールです（図2）．

さらにライブラリモジュールは，他のモジュールから参照可能なインターフェース，つまりモジュールの外部仕様となる定義(デフィニション)モジュールと，そこで定義した機能の論理を実現する実現(インプリメンテーション)モジュールのペアで構成されます(リスト2)．これによって，これまでドキュメントとしてプログラムから分離されていた外部仕様を定義モジュールに記述し，仕様に完全に一致する実現モジュールを1対1で対応させています．外部仕様（定義モジュール）に記述されていない実現モジュールの手続き，変数，定数などは，自動的に密封され中に閉じ込められ，明確な情報隠ぺいが自然に達成されます．そして重要なことは，インターフェースの変更で生じる可能性のある定義，実現モジュール間の相違を自動チェックし，発見された場合はコンパイルを中止する安全機構を備えることができるという点です．

こうして作られた各ライブラリを利用するモジュール（プログラム，ライブラリどちらのモジュールでもよい)は，その時点では見えないライブラリの実現モジュールとのインターフェースを保証する定義モジュール＝外部仕様を参照(インクルードするのではない)してコンパイルされます．たとえば，これによりシステムエンジニア(S.E.）が外部仕様をプログラム（決定)してチームの各プログラマに渡し，プログラマがその仕様にしたがって実現モジュールを記述するといった作業手順をマシンにチェックさせて確実に実現できるのです．

これに対してCは，仕様上部品の枠組み＝カプセルの構造をもっていませ

リスト3
IMPORT文，手続き名の衝突を回避した例
MODULA-2

```
(*
Title    : filter (sample)
LastEdit: 10.Jan.92
Author   : T.Suzuki
System   : LOGITECH MODULA-2 version 3.0

機能：入力ファイルから文字データを読み
      '0'を標準エラーに表示しそのほかを
      出力ファイルに書き出す

      A>filter < infile.txt >outfile.txt
*)
MODULE filter;
FROM Terminal  IMPORT KeyPressed ;
FROM InOut     IMPORT Write,Read ; (* リダイレクト機能付入出力 *)
FROM StdIo     IMPORT StdIoType  ;
IMPORT StdIo   ;

 VAR ch : CHAR ;

BEGIN
  Read(ch);
  WHILE KeyPressed() DO
    IF ch = '0' THEN
      StdIo.Write(stderr,ch); (* 手続き名の衝突をモジュール名．手続き名 *)
    ELSE                      (* で避けている                           *)
      Write(ch);
    END;
    Read(ch);
  END ;
END filter .
```

ん．現在モジュールと呼ばれる単位は，1つ1つの関数であったり，一連の関数群を書いたソースファイルであったりしますが，同じソースに書かれた関数群の確かな論理上の関係は制作者以外の人にはすぐにはわかりません．また部品間の関係を自動的にチェックする機構が弱いため，つねにインターフェース変更によるトラブルの危険にさらされています．Cでは別ファイルの関数を利用するとき「extern　プロトタイプ宣言」を使うことができますし，また使わないこともできます．externは「この名前とパラメータリストをもつ関数の本体は別ファイルにある」ことを表わしますが，そのプロトタイプ宣言と別ファイルに実際に記述されている関数名とパラメータリストのインターフェースが一致している保証はとれず，その変更時の間違いの指摘が的確に行えるとはいえません．またCのすべての関数は基本的に大域的（パブリック）な性質をもつので，情報隠ぺいには，隠そうとする関数の1つ1つにstatic宣言を書かねばならず，細心の注意をおこたれば隠ぺいは容易に破られてしまいます．

リスト4
モジュールSYSTEMの理論的なエクスポートリスト
MODULA-2

```
AX, BX, CX, DX, SI, DI, ES, DS, CS, SS, SP, BS, (* CONST *)
BYTE, WORD, PROCESS, ADDRESS,                   (* TYPE *)
ADR, SIZE, TSIZE,                               (* PROCEDURE *)
NEWPROCESS, TRANSFER, IOTRANSFER,  SETREG, GETREG, CODE, SWI, ENABLE, DISABLE,
INBYTE, OUTBYTE, INWORD, OUTWORD,  DOSCALL, EXTCALL, . . . ;
```

パブリックな関数名の衝突は，複数プログラマで行うシステム開発ではよく起きることです．MODULA-2はモジュール名がユニークであれば手続きの重複を識別できますが，Cではすべての関数名がユニークでなければなりません．たとえば100個のパブリックな手続き（関数）がある場合，Cでは100個のユニークな関数名を付けなければなりませんが，MODULA-2では10個のモジュールに10個ずつの手続きを振り分けたとすれば，最低10個のモジュール名と10個の手続き名で済ますことができるのです（リスト3）．

複数のモジュールの関係を示す明確な構文

MODULA-2では定義モジュール

COLUMN

Ada

世界最初の女性プログラマ，かのバイロン卿の娘でもあるAugusta Ada Lovelaceにちなんで名付けられたAdaは，そのロマンチックな命名の由来とは裏腹に，米国国防総省後援のプロジェクトとして開発されたプログラミング言語である．Adaはその開発の過程においては，航空管制，兵器システム，通信システム，ミサイル誘導などの巨大なシステムにおける，組み込み型システムの構築用言語として意図されていた．また，このプロジェクトには，開発に必要な特殊なプログラミング言語の数を減らし，ソフトウェアおよび開発と保守に要する費用を抑制するという目的もあった．

国防総省がプログラミング言語に必要な要件（Pascal風の文法であること，モジュール化が可能であること，リアルタイム処理が行えることなど）を提示し，各メーカーがこの要求を満たす言語仕様を考案．最終的にはフランスのCII ハネウェル・ブル社の言語仕様が採用された．1980年に最初の基準文法書が公表され，1987年には国防省はAdaを全面的に使用することを義務づけた．また，NASAの宇宙ステーションプロジェクト，ボーイング社の747-400の飛行制御プログラムにも採用されている．ヨーロッパ各国の国防省でもこれにならってAdaを採用しつつある．

プログラミング言語の系列から見ると，AdaはALGOLからPascal，MODULA-2系の特徴である強いデータ型，プログラムのモジュール化などの機能を備えた手続き型言語である．Adaにおいては，型指定は強力で，すべてのデータ型が明確に宣言され，コンパイル時にすべての結合が行われる．マルチタスクや例外処理，開発環境など本来ならAda本体とは切りはなしたほうがよいと考えられるものまですべてを取り込んだ巨大な言語仕様，風変わりな構造などから批判も多いが，ソフトウェア工学の面から見ると，あらゆる手続き型言語の集大成としてAdaのもつ意義は決して小さくはない．

に記述した機能を，そのモジュールからEXPORT（エクスポート，輸出）されているといいます（EXPORT文はローカルモジュールに使います）．そしてエクスポートされた機能を使うときは，どのモジュールからどの機能をIMPORT（インポート，輸入）するかを指示する構文を使います（リスト1，3）．これらの構文によって名前の衝突を避けたり，モジュール間の関係を明示的に示す記述スタイルを統一するので，誰の書いたソースでもモジュール関係を容易に読み取ることができ，必要なモジュールだけを的確に捜せます．MODULA-2のリンカは実際にこれを利用して，MAKEファイルなしで自動的にモジュールを検索しリンケージする機能をもっています．

これに対してCは，外部の関数がどのファイルに存在するかという情報を記述しなくてもよいので（コメントで書かれることはある），ソース作成者以外のプログラマが保守やデバッグ作業をする場合，その情報を得るために膨大なドキュメントを検索せねばならず効率を損なっています．

低レベルアクセスに必要な構文

私たちが作るアプリケーションソフトには，マシンあるいはOSの機能を利用する部分がありますが，CやPascalには仕様中にマシン依存の記述について規定がないのでマシン依存部は処理系が言語を拡張して機能を付加してきました．

しかしMODULA-2は，実用的なシステムを記述するときにマシン依存部は避けられないととらえ，言語仕様の中にハードウェア依存機能をスマートに提供する工夫をこらしています．これがモジュールSYSTEMです．

モジュールSYSTEMは，OS機能の呼び出しやレジスタ，I/Oポートのアクセス，割り込み，アドレス型，コルーチンなどの低レベルでハードウェアを直接操作するような機能をまとめて提供します．モジュールSYSTEMがインポートされたモジュールは，マシン依存部の存在を明示するので，容易に他のモジュールと区別できます（リスト2，5）．

リスト5
コルーチン，ローカルモジュール，モニタのPC-9800用サンプル
MODULA-2

```
(*
Title   : コルーチンサンプル
LastEdit: 10.Jan.91
Author  : T.Suzuki
System  : LOGITECH MODULA-2 version 3.0
*)
(*$S-*)
(*$R-*)
(*$T-*)
MODULE Coroutin;
FROM SYSTEM   IMPORT INBYTE,OUTBYTE,ADR,ADDRESS,SIZE,PROCESS,
                     NEWPROCESS,TRANSFER,IOTRANSFER ;
FROM Devices  IMPORT SetDeviceStatus,SaveInterruptVector,RestoreInterruptVector;
FROM Terminal IMPORT WriteString,WriteLn ;
FROM Display  IMPORT Write ;

PROCEDURE SetModeforTimerNo0(mode:BITSET;interval:CARDINAL);
  VAR  setmode,LSB,MSB :CHAR ;
       counter,rate    :CARDINAL ;
       cpuSpeed        :BITSET   ;
  PROCEDURE nop;
  BEGIN
  END nop;
BEGIN
  INBYTE(42h,cpuSpeed);
  IF 5 IN cpuSpeed THEN (* 8/16 MHz *)
    rate := 1997 ;
    IF interval > 26 THEN interval := 26 END;
  ELSE
    rate := 2548 ;
    IF interval > 32 THEN interval := 32 END;
  END ;
  counter := rate * interval     ;  setmode := CHR(CARDINAL(mode)) ;
  LSB     := CHR(counter MOD 256);  MSB     := CHR(counter DIV 256);
  OUTBYTE(077H,setmode);  nop;
  OUTBYTE(071H,LSB);      nop;
  OUTBYTE(071H,MSB);      nop;
END SetModeforTimerNo0;

MODULE monitor[15];                (* ローカルモジュールをモニタにした例   *)
IMPORT Write;                      (* モジュール名［定数式］で優先度指定   *)
EXPORT write;                      (* ただしこの値は処理系に依存している   *)

  PROCEDURE write(ch : CHAR);
  BEGIN
    Write(ch);
  END write;

END monitor;


VAR c         : CARDINAL ;
    WorkSpace : ARRAY[0..2000] OF CARDINAL ;

PROCEDURE coroutin ;
BEGIN
  LOOP
    IOTRANSFER(cp,mp,08H);        (* タイマ割り込みとcoroutinを結び付けた     *)
    write('*');                   (* モニタ内の手続き呼び出し                 *)
    INC(c);
  END ;
END coroutin ;


VAR cp ,mp :PROCESS ;
    p      :PROC;
    save   :ADDRESS ;
BEGIN
  c := 0;
  p := coroutin ;
  SaveInterruptVector(8H,save);
  NEWPROCESS(p,ADR(WorkSpace),SIZE(WorkSpace),cp); (* プロセスを生成した *)
  TRANSFER(mp,cp);
  WriteString('Set interval timer');WriteLn;
  SetModeforTimerNo0({5,4,2},32);
  SetDeviceStatus(0,TRUE);
  REPEAT                                           (* 割り込みを待つ *)
  UNTIL c > 200;
  SetDeviceStatus(0,FALSE);
  RestoreInterruptVector(8H,save);
  WriteLn;
  WriteString('end of samle program');
END Coroutin.
```

コルーチンによる初歩的な並行プログラミング

MODULA-2は，ユーザープログラムのなかで複数の軽量なプロセスを実現する方法としてコルーチンを用意しています．コルーチンはNESPROCESSで生成され独自の作業スペースをもち，TRANSFERで自分を中断し他のコルーチンを明示的に指定して制御を切り替えます．制御を移されたコルーチンは（前に中断した時の状態を保持しているので）切り替え時の処理を再開します．またIOTRANSFERでコルーチンの起動を外部割り込みと接続できます．この機能を使って，RS232Cの受信割り込みを実現したり，タイマ割り込みで起動するスケジューラを作りTRANSFERと組み合わせてラウンドロビン方式で動作するシステムなども実現できます．コルーチンと外部割り込みが結び付くと排他制御の問題が生じます．MODULA-2は，モジュールに優先度を指定してモニタ化できるようになっています．

強力な型チェック

MODULA-2は，異なるデータ型への代入や手続きのパラメータの相違を的確に指摘します（ストロングタイプチェック）．この言語は，データ型にデフォルト（暗黙の了解）がなく，型変換を明示的に記述させてコンパイラの保証部分とプログラマの責任部分を明確に区別します．これは一見面倒なようですが，デバッグや保守作業に有効です．Cプログラムをデバッグ，保守するときに困ることはキャストなしにデフォルトの型に異なる型のデータを代入している場合，それが制作者の意図なのか間違いなのか判定できないことです．こういった場合，制作者の意図を掴むためにアルゴリズムを丹念に追ったり，制作者の技量やコンパイラの変換規則を調べなければなりません．必ず明示しなければコンパイルできないならば，こうした意味的エラーは混入しません．

エラーチェックの点では，ウォーニングの有無がMODULA-2とCのコンパイラの大きく違うところです．MODULA-2のエラーは，構文や型違いの代入といった明らかにエラーとわかる単純なものですが，Cは言語定義の時点で存在しなかったソフトウェア工学の成果を言語拡張して取り入れ言語強化したため，その代償として多くの記述レベル（形式）ができてしまいました．ウォーニングレベルは，ある時点までは許されたのに，それ以降好ましくない形式と扱われるものを区別するために設けらたといえます．つまりCの正しい記述形式はウォーニングレベルの数だけあり，表現の統一（同じになること）を困難にしています．

インクルード文がない

MODULA-2にはインクルードがありません．それは定義モジュールとIMPORT文で，必要な情報を単一のソースファイルに明確に記述するからです．

インクルード技法は有効なソース分割技法ですが，モジュール化技法とは本質的に異なるものです．論理的に1つのモジュールを表現するソースファイルをインクルード技法で分割した場合，1つ1つのファイルは完全なソースを構成するセグメントにすぎなくなり，ソース上の完結性を失います．MODULA-2の定義モジュールとCのヘッダーファイルは比較の対象になりません．先にも述べたように，MODULA-2はIMPORT文で外部手続きのインターフェースをチェックしますが，ヘッダーファイルはあなたの書いたソースセグメントに挟み込ま

表1

		MODULA-2	C

高水準	論理	PROCEDURE Calc(VAR ans:INTEGER);
		VAR wk : INTEGER ;
		BEGIN
		ReadInt(wk,5);WriteLn;
		ans := fibonacci(wk);
		END Calc ;
			なし

高水準	呼出	VAR Max :INTEGER; Calc(Max);	

低水準	論理	PROCEDURE Calc(ans:ADDRESS);
		VAR wk: INTEGER;
		p: POINTER TO INTEGER;
		BEGIN
		p:= ans;
		ReadInt(wk,5);WriteLn;
		p^ := fibonacci(wk);
		END Calc;
			static void Calc(int *);
			static void Calc(ans)
			int * ans;
			{
			int wk ;
			scanf(" %d",&wk);
			* ans = fibonacci(wk);
			}

低水準	呼出	VAR Max :INTEGER; Calc(ADR(Max));	int Max; Calc(&Max);

表 2

	MODULA-2	C（2つの表現例）

論理	PROCEDURE Init(VAR st:ARRAY OF CHAR);
	　VAR i : CARDINAL;
	BEGIN
	　FOR i := 0 TO HIGH(st) DO
	　　st[i] := '0';
	　END
	END Init;
		static void Init(st,len)
		　char st[];int len;
		{
		　int i ;
		　for (i = 0;i<len;i++)
		　　st[i]='0';
		}
		- - - - - - - - - - - -
		static void Init(st,len)
		　char *st;int len;
		{
		　int i ;
		　for (i = 0;i<len;i++) {
		　　*st ='0';st++
		　}
		}

れるだけだからです.

Cプログラムの実際のソースサイズは，インクルードファイルのソースとインクルードしているソースを合成したサイズになります．あなたのソース記述量は少ないかもしれませんが，合成したサイズは小さいとは限りませんし，インクルードされた多くの宣言は未使用です．60ページ以上もあるWindowsのヘッダーや，ヘッダーファイルの中から別のヘッダーファイルをインクルードしているX Windowsのヘッダーを合成して完全なソースファイルを作ってみれば，毎回いかに無駄の多いソースをコンパイルしているかをご覧になれます.

パラメータのアドレス渡しの自動化

MODULA-2はPascalなどと同様に，手続き(関数)のアドレス渡しの変数パラメータを「VAR　変数名：型名」で表します．こうして引き渡される変数は，自動的にアドレスが計算され，手続き内では引き渡された変数のアドレスが固定されています.

Cは関数間で変数値をやり取りする場合，関数を呼び出す側で引き渡す変数に&を附加してアドレスにします．関数内では引き渡されたアドレスはポインタ（型付の演算可能なアドレス）となり，演算することでメモリ内を自由にアクセスして内容を破壊できます．またこうしたポインタをアスタリスク1つで区別しなければなりません.

MODULA-2も，VAR宣言のほかにアドレス指定でパラメータを受け渡す低水準の方法を提供していますが，アドレスデータはADDRESS型で，アドレス計算する関数はADR()と表現して可読性を上げ，手作業の指定による間違いの混入を押さえています（表1）.

ポインタは柔軟で強力な機能ですが，それだけ危険でもあります．MODULA-2で同様のアドレス型を用いるインタフェースは，マシンに近いきわめて限定した部分でのみ使用することが一般的です．VAR宣言で自動的にコンパイラが保証する変数アドレスを引き渡せるのですから，わざわざ手作業でアドレス指定しようとは考えません．だいたい巨大なシステムをすべてこうした低レベルなインターフェースで作ること自体無謀な企てであり，常識では考えつかないことなのです.

さらにMODULA-2の便利なパラメータパッシングの例として，配列の受け渡しが挙げられます．配列の受け渡しには解放アレイ型が，また解放配列に渡された配列の最大を求める標準関数HIGHが用意されています．Cでは配列の(先頭)ポインタが渡されるだけなので，ポインタ演算の限界を決めるパラメータを渡す必要があります（表2）.

手続き(関数)の入れ子構文

MODULA-2では，手続きの中にそこだけで使う手続きを入れ子で記述できるので，ロジックの正確な表記と，名前の衝突を抑制できます(リスト6).

リスト 6
手続きの入れ子・リスト1の手続き Calc と fibonacci
MODULA-2

```
PROCEDURE Calc(VAR ans:INTEGER);
  PROCEDURE fibonacci(VAR wk:INTEGER):INTEGER;
    VAR x1,x2,x3,i : INTEGER ;
  BEGIN
    x1:=0 ;x2:=1; i:=0; x3:=x2;
    WHILE i < wk DO
      x2:=x2+x1;x1:=x3;x3:=x2;INC(i);
    END;
    RETURN x1;
  END fibonacci;
  VAR wk : INTEGER;
BEGIN
  ReadInt(wk,5);WriteLn;
  ans := fibbonacci(wk);
END Calc;
```

手続き型

MODULA-2では，Cの関数へのポインタをより明確に手続き型と定義します．プログラム内で手続き型変数に手続きを代入し，動的束縛を実現します（リスト7）．

リスト7-1 手続き型ーフロントエンドプロセッサ制御 MODULA-2

```
(*
Title    : フロントエンドプロセッサ制御
LastEdit: 10.Jan.92
Author   : T.Suzuki
System   : LOGITECH MODULA-2 version 3.0
*)
DEFINITION MODULE FepCtrl ;
  PROCEDURE SetInputPoint(SystemLine : BOOLEAN);
  (* 漢字入力をカーソル位置か／最下行かにする *)
  PROCEDURE KanjiOn;
  (* ＦＥＰを起動する *)
  PROCEDURE KanjiOff;
  (* ＦＥＰを終了する *)
END FepCtrl.
```

リスト7-2 手続き型ーフロントエンドプロセッサ制御 MODULA-2

```
(*
Title    : フロントエンドプロセッサ制御
LastEdit: 10.Jan.92
Author   : T.Suzuki
System   : LOGITECH MODULA-2 version 3.0
*)
IMPLEMENTATION MODULE FepCtrl;
FROM SYSTEM IMPORT
  CX,AX,BX,CODE,SWI,SETREG,GETREG;

CONST
  NECInt = 0DCH;

  PushF  = 09CH;
  PopBX  = 05BH;

  Carry  = 0;

VAR NowMode : BITSET;

    getMode    : PROCEDURE (VAR BITSET) ;
    setMode    : PROCEDURE ( BITSET );
    kanjiOn    ,
    kanjiOff   : PROC ;
    setinpPoint: PROCEDURE (BOOLEAN);

PROCEDURE getCurrentMode(VAR mode:BITSET); (* モードの取得  *)
VAR  fepMode : BITSET ;
BEGIN
  SETREG(CX,242);
  SWI(NECInt);
  GETREG(AX,fepMode);
  mode := fepMode ;
END getCurrentMode;

PROCEDURE setCurrentMode( mode : BITSET) ; (* モードの設定 *)
BEGIN
  SETREG(CX,241);
  SETREG(AX,mode);
  SWI(NECInt);
END setCurrentMode;

PROCEDURE inputPoint(SystemLine : BOOLEAN);(* 入力位置を指定する *)
BEGIN
  getMode(NowMode);
  IF SystemLine  THEN
    IF NOT( 0 IN NowMode ) THEN  NowMode := NowMode + {0};  END ;
  ELSE
    IF  0 IN NowMode  THEN  NowMode := NowMode - {0};  END ;
  END;
  setMode(NowMode) ;
END inputPoint;


PROCEDURE fepOn;  (* ＦＥＰの起動 *)
BEGIN
  SETREG(CX,239); SWI(NECInt);
END fepOn;


PROCEDURE fepOff; (* ＦＥＰの終了 *)
BEGIN
  SETREG(CX,240); SWI(NECInt);
END fepOff;


PROCEDURE SetInputPoint(SystemLine:BOOLEAN);
BEGIN
  setinpPoint(SystemLine);
END SetInputPoint ;

PROCEDURE KanjiOn;
BEGIN
  kanjiOn ;
END KanjiOn;

PROCEDURE KanjiOff;
BEGIN
  kanjiOff;
END KanjiOff;


PROCEDURE NECFepEnable():BOOLEAN ; (* アプリケーションへの解放 *)
VAR result : CARDINAL ;
BEGIN
  SETREG(AX,0) ;
  SETREG(CX,224) ;                 (* NEC日本語入力機能をＡＰに開放*)
  SWI(NECInt);
  GETREG(AX,result);
  IF result = 1 THEN               (* 使用可能 *)
    RETURN TRUE ;
  ELSE
    RETURN FALSE ;
  END ;
END NECFepEnable ;

PROCEDURE NECDriverInstalled(): BOOLEAN ; (* ドライバインストールのチェック *)
VAR result : CARDINAL ;
    status : BITSET ;
BEGIN
  SETREG(CX,247);     (* NEC逐次／連文節変換ドライバの有無取得 *)
  SETREG(AX,0);
  SWI(NECInt);
  CODE(PushF);
  CODE(PopBX);
  GETREG(AX,result);
  GETREG(BX,status);
  IF Carry IN status THEN
    RETURN FALSE ;
  ELSIF (result = 0) OR (result = 3) THEN
    RETURN TRUE
  ELSE
    RETURN FALSE ;
  END ;
END NECDriverInstalled;

PROCEDURE dummyfunc0(VAR b: BITSET);
BEGIN
  b := {} ;
END dummyfunc0 ;

PROCEDURE dummyfunc1(b: BITSET);
BEGIN
END dummyfunc1 ;

PROCEDURE dummyfunc2 ;
BEGIN
END dummyfunc2 ;

PROCEDURE dummyfunc3(b: BOOLEAN );
BEGIN
END dummyfunc3 ;

BEGIN
  IF  NECFepEnable()  AND  NECDriverInstalled() THEN
    getMode    := getCurrentMode ; setMode    := setCurrentMode ;
    kanjiOn    := fepOn          ; kanjiOff   := fepOff         ;
    setinpPoint:= inputPoint     ;
  ELSE
    getMode    := dummyfunc0     ; setMode    := dummyfunc1     ;
    kanjiOn    := dummyfunc2     ; kanjiOff   := dummyfunc2     ;
    setinpPoint:= dummyfunc3     ;
  END ;
END FepCtrl.
```


PC-9801 I/O ポートマップ
機種：④⑤⑥　ポートアドレス：XXXXXXX1I0001AA0　サウンドボード　使用チップ：YM2203

アブソリュート変数の指定構文

MODULA-2は，変数を固定アドレスに割り当てられるのでVRAMや割り込みベクタなどを容易にアクセスできます（リスト8）．

リスト8
アブソリュート変数ーベクタ0Fhから8hまで内容表示
MODULA-2

```
(*
Title    : アブソリュート変数指定サンプル
LastEdit: 10.Jan.92
Author   : T.Suzuki
System   : LOGITECH MODULA-2 version 3.0
*)
MODULE ReadVector ;
FROM InOut IMPORT
  WriteHex,Write,WriteLn;

VAR  vectorTable[0000H:0040H] : ARRAY[0..15] OF CARDINAL ;
     i  : CARDINAL ;
BEGIN
  i := HIGH(vectorTable)+1;
  REPEAT
    DEC(i);WriteHex(vectorTable[i],4);Write(':');
    DEC(i);WriteHex(vectorTable[i],4);WriteLn    ;
  UNTIL i = 0;
END ReadVector .
```

開発事例による数値

MODULA-2の生産性，開発効率を示す開発事例（表3）を紹介します．経験1年未満のプログラマ1人が，平均約2540ステップ／月の生産を可能にしています．適当な言語を選択することも生産性向上への1つのアプローチではないでしょうか．

表3

開発分野	OA業務（新規開発）
メンバ	システムエンジニア(S.E.)　1名ーMODULA-2経験4年 プログラマ(P.G.)　2名ープログラム経験1年未満 他に，経験1年のプログラマ　2名が約1.5か月稼動
ファイル数 出力帳票数 プログラム数	ISAMファイル12個，マスターファイル38個 28種類． 約130本（オーバーレイ構造）
製造／テスト期間	6か月（プログラマ稼動は5か月　分析設計期間をのぞく）
ソースサイズ	約62Kステップ（コメント，既存ライブラリを除く）
記述比率	S.E.約50%　P.G.約40%　その他P.G.約10%
月産ステップ	S.E.　(63500×0.5)/6　＝約5290ステップ／月 P.G.　(63500×0.4)/5/2＝約2540ステップ／月

MODULA-2をめぐる最近の動向

標準化

MODULA-2の標準化は1984年イギリスで始まり，国際標準化のために1986年にISO／IEC JTC1／SC22／WG13（WG13）が召集されました．イギリス，カナダ，フランス，西ドイツ，オーストリア，スイス，オランダ，ノルウエー，オーストラリア，ニュージーランド，台湾からメンバーが集まり（電子立国の日本は入っていない），草案を作成しています．

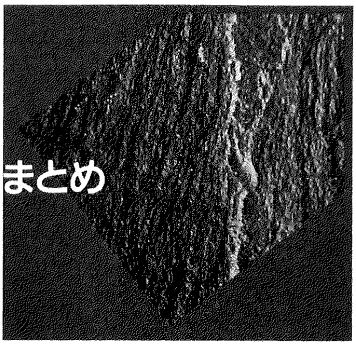
OberonとMODULA-3

「MODULA-2の子供たち」と評されるこれらの言語は，現在一方は研究分野で，他方は産業分野で実現されています．

Oberonは，ETHで開発されたワークステーションCeres-3のためにヴィルト教授が設計した言語です．OberonはCeres-3のOS名でもあり，プロジェクト名でもあります．Oberon（オベロン）という名前は，このプロジェクトの着想がヴォイジャーが天王星の月オベロンを通過したときで，その正確さに感動して命名されたというユーモラスな話が伝えられています．言語としてのOberonはどれだけ少ない語でソフトウェアを構築できるかという点から設計され，徹底した合理

化がはかられると同時に，データ型の拡張機能を付加し，オブジェクト指向を取り入れています．現在パブリックドメインソフトがMacintosh, Sunスパークステーション上で利用できるようです．

MODULA-3は，DEC System Research CenterとOlivetti Research Centerのジョイントプロジェクトとして，ヴィルト教授の賛同を得てModula-3委員会がMesa, MODULA-2, Cedar, MODULA-2+などをベースに構築した言語です．開発の基本方針として，シンプル，安全，証明済みの機能というMODULA-2のコンセプトを基礎に設計されたこの言語は，MODULA-2の言語仕様をわずかに削り，オブジェクト，例外処理，スレッド，ガベージコレクションなどの機能を大幅に追加しています．当初の計画ではSUN-3上のSunOS, VAX上のバークレイ版UNIX4.3などでリリースの予定でしたが，現在はDEC SRCが，MODULA-3 TO Cトランスレータ，Abstract Syntax Treeツールキット（Olivettiが開発），DEC, IBM, HP, Sunのワークステーション用コンフィギュレーションファイルを含むランタイムシステムをSRC MODULA-3として配布しています．



まとめ

Cプログラミングはライオンと猛獣使いになぞらえることができます．猛獣使い（プログラマ）はつねに注意深くライオン（C）を取り扱わなければなりません．気を抜くとライオンに襲われ，猛獣使いは大怪我をします．たしかにCは1人でメモリ内を柔軟に扱うプログラムを作るときは便利です．しかし2人以上でプログラムを作る場合は，ちょうどメンバ全員が1本の歯ブラシで歯を磨くようなもので，誰かが伝染病にかかったら取り返しがつかないといったリスクを背負っています．Cのプロジェクトで成功するには，アプリケーションの検討に入る前に，技量の揃った熟練メンバを集め，全員で言語自身の自由度をどこまで抑制するか綿密な規則を作り，鋼鉄の意志でそれを守ると決定することが必要です．

MODULA-2が提供する情報隠ぺいや安全性について，優秀なCプログラマはstatic宣言で情報隠ぺいはできるし，プロトタイプ宣言してウォーニングレベルを上げればチェックを厳しくできるといいます．しかし問題はまさにその“できる”にあります．プログラムの品質に直結するプログラミング言語にある人にはできるがある人にはできないという性質があることは，システム開発の生産性に大きな影響をあたえます．

MODULA-2はこうしたできる／できないの問題を，言語仕様で自動的に「なる」ようにするので，チームによる開発に望ましい言語となっています．現在世界中でWindows, OS/2, UNIX用に多くのMODULA-2の処理系が発売されていますし，幸いMODULA-2の制御構造やデータ構造は，最近Pascal化しているCと共通部分も多くあります．WindowsやOS/2のシステムコールの関数名やIMPORT, EXPORTといったMODULA-2スタイルを取り入れています．こうした状況下でMODULA-2を学ぶことは，優秀なCプログラマを短期間で育てることにもつながります．実際にMODULA-2プログラマは，Cのポインタを2週間も勉強すると，高品質なCプログラムを書きます．またそうしたCプログラムはMODULA-2スタイルを踏襲するので，ソースの標準化も無理なく実現されます．

オブジェクト指向は，構造化技法と同じく言語から発生し，最近は設計・分析の分野でも研究されています．オブジェクト指向言語（OOL）は，クラスとオブジェクトを中心にabstraction（抽象化），encapsulation（カプセル化），inheritance（継承），polymorphic reference（多態参照），dynamic binding（動的束縛）など多くのエレガントな概念を実現します．カプセル化と一部の動的束縛などを提供し，関数ベース言語とOOLの間に位置付けられるオブジェクトベース言語に分類されるMODULA-2は，オブジェクト主義者から見ると，OOLの多くの機能がなくものたりないかもしれません．しかしOOLは，開発の現場から見ると，その継承，多態参照といったエレガントな機能が新たにコミュニケーションの問題を引き起こさないか，また新人をOJTで即戦力にしなければならない現状で高機能

PC-9801 I/Oポートマップ
機種：(1)(2)(3)　ポートアドレス：XXXXXXXX1001XAA0　1MB FDDインターフェース　使用チップ：μPD765A（765相当）

なOOLの習得に投資できるかといった点で疑問を感じます.

モジュール構文による情報隠ぺいと,それらのカプセルの組み合わせ時のインターフェースチェックを厳格に行うという単純で十分に検証された技術で,中規模,大規模プログラムを安全に構築できるMODULA-2は,厳しい現状の中で使い続けることができる強力なツールであると同時に,CやOOLを使いこなすためにも最初に選択できる強力なツールであると考えます.

参考資料

「MODULA-2の歴史と目標」Niklaus Wirth著/日経バイト 1985.1
「1984年度ACMチューリング賞受賞記念講演」Niklaus Wirth/ビット 1985.6
「アルゴリズムとデータ構造」Niklaus Wirth著/浦昭二・國府方久史訳/近代科学社(この名著の名前だけを盗った類似書が多いので要注意!)
「翻訳系構成法序説」Niklaus Wirth著(第3版)/筧捷彦訳/近代科学社
「Compilerbau(第4版)」(上記翻訳原典の第4版−ドイツ語)Niklaus Wirth著/Teubner-Studienbucher
「MODULA-2プログラミング(第3版)」Niklaus Wirth著/斉藤信男訳/マイクロソフトウェア
「PROGRAMMING IN MODULA-2(第4版)」(上記翻訳原典の第4版)Niklaus Wirth著/Springer-Verlag
「Modula−2文法入門」中村和郎著/CQ出版社
「Modula−2入門」Paul M. Chirlian著/武市哲也監訳/アスキー
「Modula−2プログラミング」中村和郎監訳/マグロウヒルブック
「MODULA-2 A SOFTWARE DEVELOPMENT APPROACH」G.A.FORD, R.S.WIENER著/JOHN WILEY & SONS
「FROM C TO MODULA-2...AND BACK」C.A.WIATROWSKI, R.S.WIENER著/JOHN WILEY & SONS
「ABSTRACT DATA TYPES IN MODULA-2」R.Harrison著, JOHN WILEY & SONS
「What'New With Modula-2」K.N.King著/Dr.Dobb's Journal 1991.6
「ソフトウェアの論理的設計法」野口健一郎著/共立出版
「ソフトウェアの複合/構造化設計」G.J.マイヤーズ著/國友義久・伊東武夫訳/近代科学社
「コンパイラ・コンパイラ」P.Rechenberg, H.Mossenbock著/玉井浩訳/サイエンス社
「データ,制御構造,コンパイラ」SuperASCII 1991.2
「Modula-2によるマルチタスキング」Michael Bisignani著/ComputerLANGUAGE 1991.2/岩谷宏訳/C MAGAZINE 1991.6
「並行プログラミングの原理」M.Ben-Ari著/渡辺栄一訳/啓学出版
「平行動作プログラムの構造」P.B.Hansen著/田中英彦訳/日本コンピュータ協会
「分散システム」G.F.Coulouris, J.Dollimore著/水野忠則・福岡久雄・斉藤正史・落合真一訳/電気書院
「最近の並列分散OSとMachにみる実現技術」清水謙多郎著/講習会テキスト
「OBJECT-ORIENTED ANALYSIS」Peter Coad, Edward Yourdon著/Prentice-Hall
「OOA/OODを理解する」Tim Korson, John D.Mcgregor著/横手靖彦・福崎俊博訳/SuperASCII 1991.6・7
「組み込み応用のためのオブジェクト指向」小林史典・宮田裕之著/インターフェース 1987.1
「Oberon」Dick Pountain著/BYTE 1991.5
「THE OBERON SYSTEM(User Guide and Programmer's Manual)」Martin Reiser著/Addison-Wesley
「System Programing with Modula-3」Gren Nelson編集/Prentice-Hall
「もう一つのOOP言語Modula-3(原題:Modula-3)」サム・ハービンソン著/BYTE 1990.11/日経バイト 1991.5
「パソコン言語入門『Modula−2』」リリス鈴木/I/O 1989.4
「MODULA-2」鈴木朋夫/I/O 1989.12
「Wirth's Fourth Edition and a Standards Update」K.N.King著/Journal of Pascal,Ada & Modula-2 SEPTEMBER/OCTOBER 1989
「Whither Modula-2?」Warren Keuffel著/COMPUTER LANGUAGE DECEMBER 1989

特集 Cじゃなくてもいいじゃない

MODULA-2でアプリケーションプログラムを開発する

●インタビュー　株式会社アドミラルシステム　技術部第1課課長　堀 正明氏

本誌'91年8月号の「プログラマの仕事場」でも紹介した株式会社アドミラルシステムでは，プログラム開発用の言語としてMODULA-2を採用している．ディスク管理ユーティリティ「Newton-98」やバックアップツール「Newton Backup」などのパソコン用アプリケーションソフトの開発に携わった，同社の堀正明氏に，MODULA-2を使ったアプリケーション開発の実際について，お話をうかがった．

MODULA-2の処理系 おもに利用するのは3種類

――アドミラルシステムさんではどのような処理系を使っているのでしょうか．

少し前までは，Logitech*1でしたが，最近ではTopSpeed*2ですとか，Stony Brook*3も利用するようになりました．いまはStony Brookが多いですね．

――それぞれの処理系の特徴があれば教えてください．

Logitechは「本当の」MODULA-2なのです．Cで開発をしている人から見ると，かけ離れた世界かもしれません．TopSpeedの最新版についてはちょっとわからないのですが，Stony Brookは，makeファイルを作ったりしますし，ローレベルの記述――たとえば，INTいくつとかありますよね――そういった記述がTurboシリーズとよく似ていますから，いちばんCの雰囲気に近いかもしれません．

Logitech MODULA-2は，何もかも直接的に記述するようになっています．たとえば，AXレジスタに値をセットして，キーボードBIOSを呼ぶという処理があったとします．通常ならAXレジスタ用の変数に値をセットして，INTの処理をするprocをCALLするという形を取ります．Logitechの場合は，本当にその場でAXレジスタに値をセットして，INTを発効してしまうのです．

――現在，C言語の処理系ではMS-CにしてもTurbo Cのシリーズにしても，インラインアセンブラ機能がありますが，MODULA-2の場合はどうでしょうか．

TopSpeedとStony Brookにはありません．Logitechにはあります．でも，これがかなり変わっています．直接

```
MOV   AX,1
```

とは書けないのですが，これをマシン語になおすと，2バイトとか3バイトの16進の命令になりますね．それを直接書くことができます．

――ふだんプログラムを作成する場合には，どのような方法でローレベルのコーディングを実現しているのですか．

アセンブラで別ソースを書いてリンクすることもありますし，INT程度でしたら，普通にソース中に書くこともあります．アセンブラをやると，いろいろと支障がありますので，なるべくMODULA-2で書けるところは，MODULA-2で書いています．

＊1 LOGITECH MODULA-2 Ver3.0
＊2 TopSpeed Modula-2
＊3 Stony Brook Modula-2

PC-9801 I/O ポートマップ
機種：④⑤⑥　ポートアドレス：XXXXXXXX100110A1　GP-IBスイッチ　使用チップ：μPD7210（7210相当）

Logitechの場合は，変わった形ではありますが融通がきくので，アセンブラを使わずにすんだという面はありました．

――TopSpeedやStony Brookを利用する場合，アセンブラはMASMを使うのですか．

そうです．TopSpeedには，Top-Speedアセンブラという特別のアセンブラがあります．MASMとの互換はないようですが，アセンブラの頭にPROCと書くことができます（この機能は，MASM6.0にはあるそうです）．MASMでCのPROCを書くためには，先頭にいくつかおまじないのようなものを書かなければならないのですが，そういったものを自動的に作ってくれる機能が，TopSpeedアセンブラにははじめからついているのです．

――Newton-98やNewton Backupなどのアドミラル製品も，MODULA-2で作られたのですか．

はい，そうです．

堀正明氏

自社開発のライブラリでソフトウェア資産を再利用

――本誌の1月号でお願いしたアンケートでは，エディタはBriefを使っていらっしゃるということでしたが．

Briefというエディタは，いろいろと便利な機能があります．ちょうどCコンパイラの環境版のようなもので，中からコンパイラを呼び出したり，エラーの発生した箇所へジャンプしたり，といった操作がエディタでできるのです．本当は，コンパイラに付属の統合環境をもっと使いたいのですが，プログラムが大きくなってしまうと，使えなくなってしまうので残念です．

――一度作成したプログラムをライブラリのような形で再利用することもあるのでしょうか．

はい．ですから，ソフトウェアができるまでは非常に早いですね．プログラムを開発するのに時間がかかるというのは，原因がいくつかあると思います．ひとつは先ほどいったように，コンパイル時に型のチェックが不十分なために，予想外の結果になってしまうというもの（MODULA-2では，型のチェックが厳しいのでこういうことはあまり起こりません）．それから，プログラムを作成するたびに，毎回同じような処理を自分で作成するような作り方をしているということ．前に作ったものを再利用すれば，それなりに枯れているわけですから，それだけデバッグの手間はかかりません．それから，デバッガですね．いまはCodeViewを使っていますが，以前使っていたLogitechのMODULA-2はデバッガがよかったですよ．TopSpeedもStony Brookも，それぞれ独自のデバッガはもっていますが，デバッグ情報をCodeViewコンパチブルで出力することができるのです．したがって，CodeViewも使えます．ただ，CodeViewを使う場合には多少難があって，アドレス渡しの場合に，その中味が見られないのです．もうひとつ，MODULA-2にはBITSETというデータ型があるのですが，その値が表示できないのです．その辺りがちょっとつらいですね．

――開発マシンはIBM互換機ですか．

IBM互換機は98を買うより安いですからね．いまは98でも高いマシンは車が買えるくらいの値段です．それなら車を買ったほうがいい．開発するには486のIBM互換機です．98用のソフトの開発もこれでやってしまいます．

――Stony Brookにしても，Top-Speedにしても，IBM用の処理系ですね．98用のソフトを開発するときは，ある程度までIBM互換機で作成してから，移植するのですか．

そのときに，さきほどからお話していましたソフトウェア上の「資産」を使うわけです．98用，IBM用，J-3100用のライブラリがそれぞれ用意してありまして，それぞれインターフェースはどれも同じにしてあります．コンパイルするときに98用なのか，IBM用なのか，J-3100用なのかを指定して，条件コンパイルを行うわけです．ライブラリを作るまでは，たいへんですが，一度作ってしまえば後は楽ですから．ただし，Cコンパイラのようなカリカリのオプティマイズはしていませんの

INTERVIEW

で，プログラムサイズは多少大きくなります．でも，ハードディスクのサイズが数百MBという時代ですから，50KB，100KBのことを気にしても仕方ないでしょう．

――プログラマーズ・ページ1月号で掲載した著者アンケートの結果では，統合環境のようなものはあっても利用しない，デバッガもSYMDEBだけという方もまだかなりいらっしゃいますが，どうお考えですか．

コンピュータが好きだというだけで，プログラムを作っているわけではありませんし，企業として取り組む以上は，コストも考えなければいけません．短い時間で安くあげて，たくさん作って利益を出すほうがいいですから，コマンドラインから手間ひまかけて打ち込むよりは，環境を生かしたほうがいいですね．いまや486マシンの時代ですよ．486マシンでは，なんと一太郎とVzが同じ速さで動いてしまいます．そうなったらアセンブラで書くメリットはあまりなくなりました．

型チェックのきびしいMODULA-2 デバッグはその分楽になる

――堀さんご自身は，どうしてMODULA-2という言語を使うようになったのでしょうか．

MODULA-2がソフトウェア工学的にすぐれているからというような理由では全然ないんですよ．ただ，MODULA-2を使っている部署に配属されたというだけなのです．これについては，私がコンピュータの仕事に就いたいきさつからお話した方がいいでしょう．

もうほとんど知っている方がいらっしゃらないと思うのですが，最初はCHILLという言語を使っていました．CHILLのCはCCITTのCで，これは，電子交換機用のソフト開発専用の言語なのです．つまり，NTTの電子交換機関係のソフトを作ったのがコンピュータでの最初の仕事でした．その部署ではCHILLだけ，ほかに使ったとしてもせいぜい電子交換機のマシン用のアセンブラでした．

CHILLという言語は元々PL/Iをベースにしたもので，それにPascalを足したような感じで，Cとはやや趣が違います．この仕事を3年ほどやっていたのですが，その部署を引き上げて来て……実は私はこの会社に来る前は，日本でMODULA-2を売っていた会社の，販売してサポートする部署にいたのです．仕方ないですよね，ほかに選択の余地はないのですから，自然とMODULA-2を使うようになりました．

アドミラルシステムでは，私より前からいた方々が，こちらの方はまじめな理由でMODULA-2を使うという決定をしていました．MODULA-2を通じてこのアドミラルシステムと出会ったというわけです．アドミラルシステムがMODULA-2という言語を選んだ理由は，COMDEXとかデータショーなどを見て歩いて，いろいろな言語を比較した結果だと聞いています．

――MODULA-2という言語には，言語の仕様そのものに，並行処理の機能が含まれているそうですが，どういった内容なのでしょうか．

実際には並行処理の機能はほとんど使っていません．元々ヴィルト先生が作った機能を無理矢理パソコンに載せたという感じなのです．MODULA-2処理系の発売元では，ひとつのセールスポイントとして，デモプログラムなども作っているようですが，MS-DOSは本来シングルタスクのOSですからね．たとえば，ゲームなどでは，あちこちでキャラクタが動いて，見かけ上マルチタスクのように見えるものがありますね．ああいった感じのものを考えていただければいいと思います．一種のループですね．

並行処理を行うには，TRANSFERという手続きを使います．TopSpeedや，Stony Brookでは，まだ並行処理の機能は使ったことがないのではっきりしたことはいえませんが，LogitechではIOTRANSFER（おそらくパソコン用に後からつけ加えた機能だと思います）というソフトウェア割り込みを実行してくれるTRANSFERもありました．でも，並行処理自体は実務に耐えられるほどのものではないと思います．

INTERVIEW

――MODULA-2という言語は覚えやすい言語だと思いますか.

教育用の言語としては, Pascalがよくあげられますね. MODULA-2はPascalを元にしていますから, 覚えやすい, というよりきちんとしていますね. Cでは, コンパイルを通っても, 実行してみるととんでもないエラーが発生するということも考えられますが, MODULA-2はもっときちんとしています. ハードウェアにアクセスするといった低レベルの記述は, もちろんMODULA-2でもできますが, MODULA-2ではコンパイル時のチェックが非常にきびしいのです.

ANSIのCでは呼ぶ側と呼ばれる側で値の型が違うとはじかれるということになっているそうですが, MODULA-2でももちろんそうです. では, 何が違うのかといいますと, たとえば, ポインタを渡しますね. ポインタを渡すと, それが何を指しているポインタかという指定をすることになっています. その中の型が違っているだけでも, 全部コンパイラではじかれてしまいます. MODULA-2では, コンパイルしてリンクまで通れば, まずたいていの場合は動くということになりますから, その分デバッグが楽だともいえます. といっても, 私自身それほどCは詳しくないので, あまり偉そうなことはいえませんが…….

――最初にMODULA-2を覚えるときは, どういう形ではじめられたのですか. まったくの独学でしょうか.

そうですね. 最初に習ったCHILLも, 先輩や周りの人に聞きながら半分独学に近かったですね. その後MODULA-2に来たときは, CHILLでこう書くときは, MODULA-2ではこういう風に書けばよいという対応がついたので, 意外とすんなりできましたね.

――最近では, MS-WindowsのようなGUIが話題を呼んでいますが, MODULA-2ではWindowsアプリケーションも作成できるのですか.

Windowsアプリケーションでは, Windowsによって決められたオブジェクトというか, プロシージャの形というものがあります. ですから, 実際にプログラムを作成するときは, その形にあってさえいれば, Cだろうが, MODULA-2だろうが, Pascalだろうが, そんなことは一切関係ありません. Windowsの規約にあったコードが書けさえすればいいのです. MS-Cの代わりにTopSpeedやStony Brookを使うだけです. 後はSDKがあればいいはずなのです.

――Windowsアプリケーションを作成する場合, MODULA-2はMS-Cよりも有効でしょうか.

どの言語を使うかよりも, むしろWindowsアプリケーションを作るために覚えなければならないことというのが, 馬鹿にならないのではないでしょうか. Cでやるか, MODULA-2でやるかよりも, そこへたどりつく前にやることがたくさん出てくるでしょう.

MODULA-2によるサンプルプログラム

```
(*        プログラムの内容

          ある５桁の数字（６５５３５以下）を入力するとその逆並びの数字
          を表示するというものです。たとえば１２３４５を入力すると
          ５４３２１と表示されます。
*)
MODULE Reverse;
  FROM InOut IMPORT
    WriteString, WriteLn, WriteCard, ReadCard;

  PROCEDURE ReverseNumber;
  (* writes the digits of a nonnegative
    integer in reverse order *)
  VAR
    number: CARDINAL;

  BEGIN
    WriteString('Enter number to be reversed: ');
    ReadCard(number);
    WriteLn;

    WriteString('The reversal is: ');
    REPEAT
      WriteCard(number MOD 10, 1);
      number := number DIV 10;
    UNTIL number = 0;
    WriteLn;
  END ReverseNumber;

BEGIN
  ReverseNumber;
END Reverse.
```

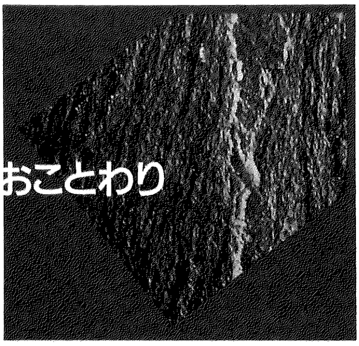
特集
Cじゃなくても
いいじゃない

生越昌己

究極のマクロ言語Lisp

Lisp vs. C

Lispは「人工知能のアセンブラ」といわれるように，人工知能関係のプログラムに利用されることが多いプログラミング言語であり，list処理を得意とし，複雑なデータ構造を扱いやすいという特徴をもっている．ここではこれらの特徴とは別に，Lispをマクロ言語として活用する方法とCプログラマがLispを学ぶメリットについて述べる．



おことわり

私はいわゆるLispプログラマではありません．ですから，この話もいわゆるLispプログラマのための話ではありません．Lispプログラマじゃない私にそのへんの話を期待するほうが無理というもんです B)　そのつもりで読んでください．私の本業で普段使っている言語は，C or COBOL-85です．

じゃ何でそんな奴がLispの話なんか書くんだい？　と聞かれそうですね．これは編集部からのお願いで……もあるんですけど，「C vs Lisp」というお話を期待されてのことです．「Cプログラマから見たLisp」みたいな話でしょうね．

と書くとLispはまったく素人のように聞こえてしまいますが，かつてはあるところで人工知能関係の仕事をしておりましたし，そのへんの趣味もあってけっこうLispコードは読んだり書いたりしておりました．知識ベースとか言語理解が主ですけどね．でも入門は数式処理なんです．

常識的Lisp像

Lispとは――いわずと知れたlist prosessorですね．つまりはlist処理を行うための処理系．listといっても「ソースリスト」とかのlistじゃなくって，ポインタで繋がったデータ構造のほうのlistである……まぁこのへんは情報処理試験に出るくらいの常識のコンコンチキ．実際にどんなところで使われているかといえば，まぁ最初に思い浮かぶのはAI――いわゆる人工知能って奴ですね．何といっても

人工知能のアセンブラ

と呼ばれることもあるくらいですし，ちょっと前のLisp処理系の広告なんかには必ず「人工知能言語」なんて書いてありましたよね．じゃなぜLispは人工知能関係のプログラムに使われるかといえば，「複雑なデータ構造が容易に扱える」からですね[1]．「複雑なデータ構造」ということだけなら，CでもPascalでもがんばって書けば，書いて書けないことはないですし，実際最近の実用知識シェルなんかはCで記述されているのですが，研究用や試作しているシステムではデータ構造はコロコロ変わりますし，ものによってはデータ構造を変形するタイプのシステムもありますから，Cのような言語では記述が困難になるわけですね．このへんのLispのさまざまな柔軟性は非常にありがたいことです．Cで書いても書けないことはないでしょうが，それを行うにはLispの処理系を書くのと同じくらい手間の掛かる話です．

とまぁこのへんの話は，普通のLispの教科書にも書いてある話ですね．これをもってして，Lispのほかの言語に対する優位性を説くことも可能ですが，このへんの話はほかの人のほうが得意そうですし，インターフェース誌にでも任せておけばよいので，ほかの「Cプログラマうけするメリット」を探してみましょう．

Lispは徹底した関数言語である

これもよくいわれる優位性です．このへんの議論の展開は理論畑の人に多くて，純粋な理論方面の研究がされていたり，理論畑の人からは「Lispは完全純粋な関数言語ではない」といわれたり，マクロだぁ汎関数だのfunarg問題だのとむずかしい議論の展開が始まります．まぁ元々Lispという言語の枠組みがλ算法を基本としているために，この方面の議論が盛んに行われたり，また限られた範囲での議論では純粋な関数言語（λ算法言語といいかえてもよい）であるという点や，また細かい部分を無視すればかなり現実的な問題に至るまで関数言語としても展開やデバッグの方法論の適用が可能ですから，この方面の議論や優位性が主張できるようですが，私にはさっぱりわかりませんから解説不可能です．興味のある方はbitだのコンピュータソフトウェアだのを読んでみればよいでしょう．

さてここまで書いておいて本稿でのLispに関するアプローチを書きますと，

Lispは非常に強力なマクロ言語である

この側面についてお話ししていきたいと思います．list処理や関数言語なんていうのが必要でない人でも，マクロ言語には用もあるでしょうし，Cプログラマもこの点については興味があるでしょう．

＊1)ほかにも理由はありますが……．

ここでいうところの「マクロ言語」とは，おもに「コマンドの拡張用に用意された言語」という意味です.「Vzのマクロ」とか「TeXのマクロ」とかという奴です．皆さんはマクロ言語って好きですか？　私はあんまり好きではありません．というのもいわゆるプログラミング言語と違って，制約が多かったり柔軟性に欠けていたり，わけがわからないくらい節操がなかったり……，プログラミング言語と比べると，ずいぶん「言語」として未完成な感じがします．また多くのマクロ言語というのは，言語仕様の関係でしょうか「黒魔術」的な技術を必要としますしね．なんたって「まっくろ」B)

前節でも書きましたが，世間一般にいわれているLispのメリットというのは，Cプログラマにとってはあまり用のないことが多いように思います．Cで書くようなlist処理というのは，まぁ普通いちばん複雑な例でもコンパイラの最適化フェーズ程度でしょうし，普通はあまりlist処理にならないように設計するのが本来でしょうから，たかが知れています.またCはLisp同様に「関数型言語」と呼ばれることもありますが，それでどうこうということはあまりありませんし，このことがC本来の使い方に影響を与えるとは思えません．Lispで行われている理論的研究の成果を使う……ということも考え難いことです.またLispに関するさまざまな悪評

①構文が不気味
②実行が直接見えない
③遅い

なんていうのがありますから，高速簡潔を家訓とするCプログラマにとっては，あまり魅力のない言語だと思います.まぁ逆にLispプログラマからCを見れば，

①構文要素が複雑
②記憶管理がやっかい
③低機能

なわけですから，おたがい様というところでしょうかB)

ということで「まぁ住んでる世界が違うんですね」なんていってしまえばそれまでの話．実際に多くのCプログラマやLispプログラマがそう思ってきたわけなので，たしかに住んでいる世界が違うといえばそうなんですが，それではあまりに視野が狭過ぎますね．でもよく両方の文化を考えてみると，いずれもhackerを産み出した言語でありますから，似ている面もあるんでしょうね．

Cプログラマの多くは，言語の選択として「まずCで書いてみる．もし操作上の困難や速度上の問題があればアセンブラで書き直してみる」ってアプローチをすると思います．かつて

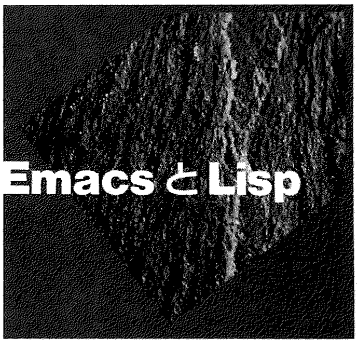
Cは高級アセンブラ

と呼ばれていましたが，結局どうやら汎用言語としての地位を獲得したようです．なんたって，

Cに書けないプログラムは存在しない

と思えるくらいあちこちで使われていますし，ライブラリを整備すれば「高級アセンブラ」なんて思わないくらいアプリ寄りの記述も可能になってきています．とくにC++なんかはそうですね．

そんな背景で果たしてLispなんかを採用したり，知識としてもっている必要があるでしょうか？　まぁいままで知らないで過ごしてきた人も多いでしょうし，これからも知らないで過ごすことも可能でしょうから，ないといえばないといえないこともないでしょう．それでも敢えて私は「知ってて損はない」といいましょう．理由は前節でちらっと書いたように「マクロ言語としてのLispの側面」と，もう1つは「教養」です．



ところで，世間でいちばんたくさん動いている実用Lispシステムは何でしょう．これはおそらく「Gnu Emacs」ではないかと思います．ご存じのように，Gnu EmacsはRichard M. Stallmanの作ったcopyleftなU.＊X上のテキストエディタで，非常に豊富な機能と事実上無限といってよい程の拡張性をもっている「プログラマの理想郷」といってもよいようなものです．そしてまたその構造が，「Cで書かれたエディタの基本部分とLisp実行系の上にLispでコマンドが実装されている」ということもご存じだと思います．つまりそれを意識して

PC-9801 I/O ポートマップ
機種：(1)(2)(3)　ポートアドレス：XXXX00XX1010AAA1　文字パターンROM

いるかどうかは別にして，多くのCプログラマがお世話になっているLispシステムなわけです．最近はMS-DOSにも移植されましたから，DOS上で開発している人も使っていることが多いんじゃないでしょうか．

このGnu　Emacsはいうなれば「LispとCのオイシイ関係」でありまして，両方のメリットを十分に引き出したシステムだと思います．たとえば，パフォーマンスを要求されるエディタの基本操作にあたる部分はCで記述されていますし，高機能なコマンドの多くはLispでスッキリ書かれています．また，U.＊ XやDOSの多くのCではプログラムを実行しながら拡張を行うなんていうことは逆立ちしてもできない相談ですが，コマンド言語にLispを採用している関係から，自分自身を実行しながら拡張することは容易ですし，またそれを支援するための機能を豊富に備えています．そしてそのLispの処理系はCで書かれているために移植性も高く，Emacs Lisp自体はLispとしてかなり変態的なのですが，「某Lispがないとこのアプリは動かない」ということはなく，処理系ごと移植する……ということが可能になっています．このへんはLispのもつ第3の側面である「強力なマクロ言語」という面がいかんなく発揮されている部分です．

ある通信ソフトとの出会いと失望とTmacs

Gnu Emacsほどではないですが，このLispをマクロ言語としてもつシステムがけっこうあります．たとえばAuto-CADなんかはそうですし，メガソフトの通信ソフトであるMiLinkはマクロ言語がSchemeだそうです．また私がXLispというfreeなLisp処理系をベースに作った通信ソフトもマクロ言語がLispになっています．以下この自作通信ソフト（Emacsのマネをして Tmacsと呼んでいます）の話をベースに「マクロ言語としてのLisp」についてお話ししていきましょう．

Scheme風マクロ言語をもつ通信ソフト

いきなり話が横道にそれますが，私はパソコン通信のネットの1つである日経MIXに参加しています（この原稿もMIXで副編集長からメールで頼まれてしまったものです）．私の住んでいるところは島根県松江市――わからない人もいるでしょうね．出雲大社の近所，竹下元首相の選挙区……早い話

COLUMN

Logo

Logoは，1960年代にアメリカのボルト・ベラック&ニューマン（BB&N）社をスポンサーとしてマサチューセッツ工科大学（MIT）の研究者グループによって開発された．Logoプロジェクトの目標は，タイムシェアリングシステムと対話型コンピュータ言語を組み合わせて，子供たちに数学の基本概念を教える方法を探るというものだった．このプロジェクトには，人工知能の専門家からプログラミング言語の設計者，教育学者，心理学者までが参加していたが，そのなかの1人であるシーモア・パパートは，子供たちが遊びのなかから知識を得る方法，つまり「ものごとの仕組みについて心のなかで仮説をたて，それがまちがっているとわかると仮説を修正して新たな知識を得る」ことをコンピュータとプログラミング言語を用いてディスプレイ上に作り出せると考えていた．

彼らは子供たちを抽象的な思考に慣れさせるために，ディスプレイ上に三角形として示されるタートル（亀）を作り出した．タートルは，子供たちの命令にしたがってディスプレイ上に線を引き，幾何学図形を描き出す．子供たちは，このタートルに言葉を教え，命令し，図形を描かせることを通して数学の基本概念を身につけていくことができた．

Logoの文法はLispに準拠しているが，これは本格的なプログラミング言語と同等に柔軟な思考を可能にしながら，子供たちが自力で探求できる単純さを実現するために拡張性に優れたLispが合っていたからだ．Lispを基盤としているために，Logoではデータをおもにリストとして扱っている．また，やはりLisp同様にリスト自体をリストの構成要素にすることが可能である．さらに，Logoにはワードと呼ばれる独自のデータ型がある．ワードはLispのアトムの概念に近いもので，文字列処理の部分が強化されている．

これらのことからLogoは子供に学びやすいだけでなく，複雑なリスト処理プログラムを実現することもできる高機能な言語になっている．現在ではLogoは，全米の小・中学校の教育用プログラミング言語としてカリキュラムに取り入れられている．ちなみに，Logoとはギリシャ語で「思考」を意味する言葉である．

が東京からうーんと離れた僻地です．ここから東京にあるホストに入るのですから，通信料はたいへんなものです．ですから普通のダム型の通信プログラムで入ってトロトロとコマンドを打っていたり，書き込みをしていたのではほんとうに死んでしまいます．また，書き込みは多くの人の目に触れるものですから，十分推敲してからホストにアップしたいものです．また「パソコン通信」に限らずコミュニケーションというのはその内容自体に意味があるのであって，手段はどうでもよいことですね．つまりホストのコマンドなんていちいち考えたくないわけです．

というさまざまな要求を満たすために自動運転のできる通信ソフトが欲しかったわけです．噂に聞くところによると，CCTやULTなんかは非常に強力なマクロ言語が用意されていて，それこそ「何でもできる」らしいのですが，あいにく私の通信で使えるパソコンはド・マイナー機であるFMRです．最近はULTminiが移植されているのですが，私が「欲しい」と思っていた当時はまだ移植されていませんでした．で，ULTでもほかの通信ソフトでもそうなんですが，マクロはとても言語といえたしろものではありませんよね．一見言語なんだけど，何かミョーな癖というか変な部分があったり，現代の構造化言語から考えると信じられないような構文だったり．そこでたいがい諦めてPascal風言語をもった通信ソフト（Pascalの処理系作成は得意ですB）を設計しようと思っていたところに，「某NiftyにLisp風マクロをもった通信ソフトがあったよ」と教えてくださった方がありました．早速入手してあれこれ調べて見ました．それでわかったことは

①言語はScheme風
②scopeはdynamic（Schemeはstatic）
③通信に必要な機能はたいてい備わっている
④ファイル操作機能が貧弱

といったところです．言語がScheme風なのに，scopeがdynamicなのは変だなぁとは思うのですが，これは用心して使いさえすれば，まぁ実害はありません．それよりもファイル操作機能が貧弱なのが問題です．なぜなら，私の希望しているオートパイロットは，単にホストに接続してメッセージを読んでくるだけではなくて，自分の書き込むメッセージを自動でアップロードして欲しいのです．何日かこのへんを解決する方法を考えたり，処理系を拡張することを考えたりしたのですが，どうにもうまくできる方法を考えつかなくて（トンチみたいな方法はあるのかもしれませんが，私はせっかく「言語」を使うのだから正攻法でやりたいと思いました），採用は諦めました．そして「この程度なら元の処理系さえあれば数日で作れる」と踏んで自作することにしました．これが現在Tmacsと呼んでいる「Lispをマクロ言語としてもっている通信ソフト」の誕生です．

XLisp Ver 2.0ベースの通信ソフトの開発

オートパイロット付き通信ソフトに必要な基本機能を考えてみると，以下のことがあげられると思います．

①通信ポートの制御（あたりまえ）
②受信文字（列）の検出操作
③時間監視

逆にいえば，これらの機能があれば通信ソフトとしてのさまざまな機能は，すべてマクロとして記述できます（マクロ言語が言語として十分な機能をもっていれば）．

私はここで「マクロ言語」としての言語にXLispを採用しました．これは最初のほうからいっているLispの強力な言語としての機能や，インタプリタ言語のもつ柔軟さ，またXLispは処理系が公開されているとかの利点からです．自分で言語をデザインされた方ならわかると思いますが，言語の仕様や実装は技術や馬力のほかに「センス」としかいいようのない「バランス感覚」が必要とされるものです．この点Lispはすでに広く使用されている言語ですし，処理系そのものはすでに完成されたものが入手可能ですから実装技術が要求されることも少なく非常に都合のよいものです．

強力なマクロ機能をもつ通信ソフトの完成！

かくして，マクロ言語の仕様は自動的に決定しました．XLispそのものがマクロ言語となります．これに通信ソフトとして追加した機能は，だいたい前述のとおりですが，あらためて整理して書いておくと，

①通信ポートの制御
ポートのopen close，各種設定
②文字（列）の検出操作
送信受信（ポートはファイルとして扱われます）
特定の文字列受信の検出
送信受信入力出力した文字の評価
③時間監視
タイマによるwait

PC-9801 I/Oポートマップ
機種：④⑤⑥　ポートアドレス：XXXXXXXX1011AAAA　RS-232C拡張インターフェース　使用チップ：μPD8251（8251相当）

このうちで,「送信受信入力出力した文字の評価」というのは，256個の要素をもつベクタを用意しておいて，文字に対応したベクタの要素をeval してやる……ということです．ベクタの要素が関数ならその文字コードに対応する関数が実行されますし，数値なら文字コードがその数値に変換されます．このベクタを活用することによって，各種文字コードへの対応やキーへのコマンド割り付けとかいろんな応用が考えられます．まさにEmacs風の操作の実現ですね．またある種のエスケープシーケンスとかを定義しておいて，受信時にそれを実行するとか……，楽しい応用が考えられます．図1は私の現在使っている設定です．

XLisp自体の改造ステップはほんのわずかです．オリジナルに追加した関数は表1にあげただけです．たったこれだけの追加でXLispが非常に便利なマクロ言語を備えた通信ソフトに早変わりするわけですから，Lispのマクロ言語の素性のよさを感じさせるよい例だと思います．

とここまで書くと，バリバリのCプログラマの人は「なんだ同じことはCに通信用ライブラリを追加してもできるよ」といわれるでしょうね．たしかにそれはそうでしょうし,「完成した」ソフトを使うときにはむしろCでやってしまうほうがよい結果を生むと思います．しかし，通信用のマクロを書いたことのある人ならわかると思いますが，通信用のマクロというのはどんどん変化していきますし，またどんどん複雑化していくものです．マクロ言語をコンパイラ言語にするのは，ちょっと拡張に不便でしょうし，またインタプリタ言語でもBASIC（それも昔のような言語仕様の奴)のような言語を使っていると，複雑になっていくとそ

```
;;
;;  コマンドのキーバインド
;;
(defun bind-key (num body)
    (setf (aref *console-input-hook* num) body))

(bind-key 2     '(send-break 10))       ;;  CTRL-B(break 信号の送出)
(bind-key 4     '(ctrl-log))            ;;  CTRL-D(ログ取得の制御)
(bind-key 5     '(medit))               ;;  CTRL-E(エディタを使う)
(bind-key 17    '(term))                ;;  CTRL-Q(手動モードへ移行)
(bind-key 20    '(up-try))              ;;  CTRL-T  B)
(bind-key 21    '(mix-up-command))      ;;  CTRL-U(手動モード時に upload する)
(bind-key 26    '(progn
                    (end-term)
                    (top-level)))       ;;  CTRL-Z(通信をやめる)

;;
;;    内部形式では各行は'\r' のみで区切られるので、'\n' は無視する
;;
(setf (aref *port-input-hook* 10)   0)  ;;  CTRL-J
```

図1
ベクタの定義例

表1　XLispに追加した関数

(hit-keyp)	この関数呼出時にキー入力があればその文字コードが，なければ0が返ります．
(send-break n)	ブレーク信号 n＊10ms 送信します．省略すると，100を指定したとみなします．
(term)	端末動作ループです．以下のグローバル変数を参照します． - 文字フック - ＊local-echo＊ ローカルエコーのフラグ 呼び出す前にポートのオープンが必要です．
(log-open ファイル名)	ログファイルをオープンし，ログ取得を開始します． ファイル名を省略すると，xlterm.log を指定したと見なします． 失敗すると NIL 成功すると t を返します．
(log-close)	ログファイルをクローズします．
(wait-for 文字列リスト)	文字列リストは入力を待つ文字列のリスト 文字列リストに指定された文字列が入力されるまで，プログラムの実行を停止します．受信は継続されます． 文字列リストに指定された文字列が入力された場合，リストの何番目かという値を返します．先頭は0です．
(wait-time n)	n/10秒，プログラムの実行を停止します．受信は継続されます． 指定しない場合，n＝1とみなします． 待っている間に，キー入力があればその時点で復帰し，NIL を返します． 所定の時間待つと，t を返します．
(read-string n)	指定した文字数だけ文字を受信し，文字列にして返します． 指定しない場合は，..文字か＼r を受信するまでとなります．
(open fname &key :direction)	fname に以下の規約で作られた文字列を指定すると，ポートのオープンとなります．

COM:port:PCSX:baud:size:time

名前	意味	値	デフォルト値
COM	---	固定	---
port	ポート番号	0-2	0
P	パリティ	E/O/N	N
C	文字ビット長	7/8	8
S	stop-bit	1/2/0	1
X	XON/XOFF	X/N	X
baud	ボーレート	75-19200	1200
size	バッファサイズ	16-32767	1024
time	送信タイムアウト	(ms)	100

ストリームを返します．オープンされたストリームは他のファイルと同様に使用可能です．

れこそ「もずく」のようなマクロになってしまうでしょう．

実際に私が日経MIXにアクセスするときに使っているマクロの例をリスト1にあげておきます．処理としては「会議やメールを全部読んで，メッセージをアップする」というだけのものです．

マクロ言語としての Lisp の可能性

さてずいぶん長々と自分の作ったプログラムの話をしてきましたが，Lispのマクロ言語としての可能性を感じていただけましたでしょうか．元々言語として非常に強力な機構をもっていますから，最初にLispの処理系をもっていればほんのちょっと目的とするアプリケーションに必要な機構や関数を追加してやることにより，Lispがそのアプリケーションのマクロ言語となる……何か順番が逆なような気さえしますが，目的とするアプリケーションも割合簡単に作成できて非常によいものです．

ごく個人的な趣味をいわせていただくなら，Common Lispは好きではありません．言語仕様が大き過ぎますから，とても自分で実装しようという気は起きませんし，使うのも決して容易ではありません．いわゆる生っ粋のLispプログラマから見ればCommon Lispはよい環境なのでしょうが，「普段はCで書いていて，必要があればLispを使う．できればおたがいリンクして使えればいいな」なんて思ってる日和見プログラマにしてみると，Common Lispはちょっと使いにくいものです．第一，大きすぎて「マクロ言語」には使えそうにありません．

またCプログラマにしてみれば，すでにCという強力なコンパイラ言語をもっています．ですからほんとうに速度が必要であればCを使うでしょうから，Lispにコンパイラはあまり必要ありません．それよりはCにない柔軟な開発のできる点がLispの大きな魅力ですから，やっぱりLispはインタプリタに限りますね．それもCommon Lispのように巨大な処理系じゃなくって，必要にして十分な機能が軽々と動く……という処理系が望まれます．いろんな要件を趣味と独断と偏見で処理すると，Common Lisp風を期待するならXLisp．別にCommon Lisp風であることがどうでもよければSchemeってところがよいでしょうねB）あくまで主観的意見ですが，Lispを「強力なマクロ言語」として見るなら妥当な選択でしょう．もちろん，

COLUMN

Scheme

現在Lispは，記号処理分野，とくに人工知能・数式処理・定理証明，そして機械翻訳などの分野で幅広く利用されている．これらはリスト処理機能によるところが大きい．もちろんリスト処理だけでなく数値計算やグラフィックスなどの分野においても，そのシステムの生産性の高さが注目されている．

このLisp標準化の中から派生した，もうひとつのLisp，それがSchemeである．

Schemeは1970年代後半にG.J.Sussmanによって設計された言語で，Lisp 1.5から出発したCommon Lispが次第に手続き型リスト処理言語として意識されはじめたのに対し，ラムダ計算に基づいた言語としてLispのセマンティックスを考え直して作られたものである．Schemeの仕様は3回にわたる改訂を経て，現在は「Rees & Clinger 86」にまとめられ，出版されている．

SchemeがCommon Lispと大きく違う点は，サイズ，その言語のコンパクトさであるといえる．仕様書を比べてみても，Common Lispが400ページ以上あるのに対し，Schemeは50ページほどである．これならパソコン上でも十分に利用することができ，教育用に適しているといえる．とくに日本のように，教育用のワークステーションが少なく，学生がLispに直接触れる機会が少ない環境では大きな力になるだろう．

Schemeシステムの基本的動作は，プロンプトを表示し，フォームを1つ読んで，それを評価し，結果を表示するという基本的にLispと変わらないものだ．フォームとは，リストフォームと呼ばれる一連の関数やマクロの呼び出し，変数やデータなどを指す．

関数呼び出しはSchemeのもっとも基本的な制御構造で，関数は自分自身を再帰的に呼び出すことができる．また繰り返しの機能としてdo式が用意されている．

標準のSchemeではプログラム作成のための機能はまったく用意されていないが，ほとんどのSchemeにはこの機能が拡張されている．

リスト1

Lisp

```
;;;;
;;;;  global variables
;;;;
(setq *fp* nil)
(setq *local-echo* nil)
(setq *log* nil)
(setq *editor-command*  "fe -I -L -B -* ")
(setq *max-redial*  5)
(setq *my-trap* '("ogochan" "おごちゃん"))
(setq *check-try* nil)

;;
;;  内部形式では各行は'\r'のみで区切られるので、'\n'は無視する
;;
(setf (aref *port-input-hook* 10)   0)       ;;  CTRL-J

;;;;
;;;;  stuff function
;;;;
(defun open-term (file)
    (setq *fp* (open file)))

(defun close-term ()
    (close *fp*)
    (setq *fp* nil))

(defun end-term ()
    (close-term)
    (end-log)
)

(defun start-log ()
    (log-open *log-file*)
    (set-color *yellow*)
    (setq *log* t)
)

(defun end-log ()
    (set-color *white*)
    (log-close)
    (setq *log* nil)
)

(defun ctrl-log ()
    (if *log*
        (end-log)
        (start-log)
    )
)

(defun transmit (s)
    (princ s *fp*)
)

(defun op ()
    (open-term *term-file*)
)

(defun medit ()
    (end-log)
    (system (strcat *editor-command* *log-file*) )
)

(defun at-command (com)
    (transmit  com)                    ;;  hand shake
    (wait-for '( "OK\r" ))
    (wait-time 10)
)

(defun dialing (dial)
    (let*   ((ret nil))
        (at-command  "ATZ\r")
        (at-command  "ATX\r")
        (at-command  "AT&L0\r")
        (at-command  "AT\\N3\r")
        (at-command  "AT%C1\r")
        (at-command  "AT\\J2\r")
        (transmit  dial)
        (transmit  "\r")
        (let*   ((w (wait-for '( "CONNECT" "NO" ))))
            (wait-for '( "\r" ))
            (setq ret (= w 0))
        )
        ret
    )
)

;;;;
;;;;  手動用コマンド関数
;;;;

;;
;;  手動時にアップロードするコマンド用
;;
(defun mix-up-command ()
    (let*   (fn)
        (write-str "FILE NAME:")
        (setq fn (read-line))
        (mix-up-file fn)
        (write-str "FINISH\n")
    )
)

;;
;;  try 会議オンライン書き込み用
;;
(defun up-try ()
    (system (strcat "fe -I -L try.msg"))
    (mix-up-file "try.msg")
    (transmit  "A\r")
)

;;;;
;;;;  オートパイロット用関数群
;;;;
```

```
;;
;;  ログインを行う
;;
(defun mix-login-sequence ()
    (wait-for '( "):" ))          ;;  wait prompt
    (transmit  "mix\r")
    (wait-for '( "Name?" ))
    (transmit  *mix-login-name*)
    (transmit  "\r")
    (wait-for '( "word:" ))
    (transmit  *mix-password*)
    (transmit  "\r")
)

;;
;;  ログアウトを行う
;;
(defun mix-logout-sequence ()
    (transmit  "bye\r")
    (wait-for '( "back soon." ))
)

;;
;;  アップロードするファイルのリスト作成
;;
(defun up-list ()
    (system "ls *.up > upfiles")
    (setq *up-file-list* nil)
    (let*   ((ups (open "upfiles" :direction :input)))
        (do*    ((rec (read-line ups) (read-line ups)))
                ((null rec))
            (setq *up-file-list* (cons rec *up-file-list*))
        )
        (close ups)
    )
    (system "del upfiles")
)

;;
;;  実際のアップロードを行う関数
;;
(defun mix-up-loop (fp)
    (do*    ((rec (read-line fp) (read-line fp)))
            ((or (null rec)
                 (string= rec ".")))
        (transmit  rec)
        (transmit  "\r")
        (wait-for '( "\r" ))
    )
    (transmit  ".\r")
)

;;
;;  ファイルのアップロードログ取得なし
;;
(defun mix-up-file (fn)
    (let*   ((fp  (open fn :direction :input)))
        (mix-up-loop fp)
    )
)
```

```
;;
;;  メールをアップロードする
;;
(defun mix-upload-mail (fp to)
    (transmit  "mail\r")
    (wait-for '( "\rH:" ))
    (end-log)
    (transmit  "to ")
    (transmit  (string-downcase (str-cdr to #\:)))
    (transmit  "\r")
    (wait-for '( "\r 標題を入れて下さい:" ))
    (transmit  (read-line fp))
    (transmit  "\r")
    (mix-up-loop fp)
    (wait-for '( "\rsend/action:" ))
    (transmit  "send\r")
    (wait-for '( "送信してます... メモ " ))
    (let*   ((msgno  (str-car (read-string 7) #\ )))
        (wait-for '( "\rH:" ))
        (start-log)
        (transmit  msgno)
        (transmit  "\r")
    )
    (wait-for '( "\rH:" ))
    (transmit  "quit\r")
)

;;
;;  会議にメッセージを書き込む
;;
(defun mix-upload-news (fp to &optional (download nil))
    (let*   ((rec (read-line fp))
              msgno
            )
        (end-log)
        (transmit  "j ")
        (transmit  (string-downcase (str-cdr to #\:)))
        (transmit  "\r")
        (wait-for '( "R:" ))
        (let*   ((subject (string-left-trim
                              " "
                              (str-cdr rec #\:))))
            (if (char= (char subject 0) #\#)
                (progn
                    (transmit  "com ")
                    (transmit  (subseq subject 1))
                    (transmit  "\r")
                )
                (progn
                    (transmit  "say\r")
                    (wait-for '( "\r 標題:" ))
                    (transmit  subject)
                    (transmit  "\r")
                )
            )
        )
        (mix-up-loop fp)
        (transmit  "A\r")
        (wait-for '( "... メッセージ " ))
        (setq msgno (str-car (read-string 5) #\ ))
```


PC-9801 I/O ポートマップ
機種：①②③　ポートアドレス：XXXXXXXX1101XXX0　未使用

```
        (wait-for '( "\rR:" ))
        (start-log)
        (if download
            (progn
                (transmit  msgno)
                (transmit  "\r")
                (wait-for '( "\rR:" ))
            )
        )
        (transmit  "quit\r")
    )
)

;;
;;  指定したコマンドを発行する
;;
(defun mix-upload-com (com)
    (transmit  (string-downcase (str-cdr com #\:)))
    (transmit  "\r")
)

;;
;;  会議と発言番号を指定してメッセージを読む
;;
(defun mix-download-news (com)
    (let*   ((conf (str-car (str-cdr com #\:) #\#))
             (msgn (str-cdr (str-cdr com #\:) #\#))
            )
        (transmit "j ")
        (transmit conf)
        (transmit "\r")
        (wait-for '( "\rR:" ))
        (transmit msgn)
        (transmit "\r")
        (wait-for '( "\r\rR:"
                     "<リターン> を押して下さい。\rR:"))
        (transmit "quit\r")
    )
)

;;
;;  プロンプト待ちの処理、確認にはYと答える
;;
(defun mix-wait-prompt (prp)
    (let*   (prompt)
        (cond   ((listp prp)
                    (setq prompt (cons "\010" prp)))
                (t
                    (setq prompt (list "\010" prp)))
        )
        (if (= (wait-for prompt) 0)
            (progn
                (transmit  "Y\r")
                (wait-for prompt)
            )
        )
    )
)

;;
;;  アップロードファイルを解析してコマント発行する
;;
(defun mix-upload (fn)
    (let*   ((fp  (open fn :direction :input)))
        (do*    ((rec (read-line fp) (read-line fp))
                 com
                )
                ((null rec))
            (setq com (string-left-trim
                           " "
                           (string-upcase (str-car rec #\:))))
            (cond   ((string= com "TO")
                        (mix-upload-mail fp rec)
                        (mix-wait-prompt "\r:")
                    )
                    ((string= com "NEWSGROUP")
                        (mix-upload-news fp rec t)
                        (mix-wait-prompt "\r:")
                    )
                    ((string= com "UPLOAD")
                        (mix-upload-news fp rec nil)
                        (mix-wait-prompt "\r:")
                    )
                    ((string= com "READ")
                        (mix-download-news rec)
                        (mix-wait-prompt "\r:")
                    )
                    ((string= com "COMMAND")
                        (mix-upload-com rec)
                        (mix-wait-prompt "\r:")
                    )
            )
        )
        (close fp)
    )
    (system (strcat "ren "
                    fn
                    " "
                    (subseq fn  0 (strchr fn #\.))
                    ".OK")
    )
)

;;
;;  アップロードリストにしたがってアップロードする
;;
(defun mix-upload-sequence ()
    (wait-for '( "\r:" ))
    (transmit  "clear\r")
    (wait-for '( "\r:" ))
    (do*    ((w *up-file-list* (cdr w)))
            ((null w))
        (mix-upload (car w))
    )
)

;;
;;  会議メッセージのダウンロード
;;
(defun mix-auto-conf ()
    (transmit  "\r")
    (do*    ((com '( "\r 読んでいないメッセージはありません。"
```

```lisp
                    "\r 新しいメッセージはありません。"
                    "\rR:"
                    "\r:" ))
            (fexit nil)
            (w 0))
           (fexit)
        (setq w (wait-for com))
        (cond   ((= w 0)    (setq fexit t))
                ((= w 1)    (setq fexit t))
                ((= w 2)    (wait-time 5)
                            (transmit  "clear\r")
                            (wait-for '( "\rR:" ))
                            (transmit  "file all\r")
                            (wait-for '( "\r:" ))
                            (transmit  "show scr\r"))
                ((= w 3)    (transmit  "\r"))
        )
    )
)

;;
;;  メールのチェック
;;
(defun mix-auto-mail ()
    (transmit  "mail\r")
    (wait-for '( "\rM:" ))
    (transmit  "\r")
    (do*    ((com '( "\r 受信メールボックスは空です。"
                    "\rM:"
                    "\rread/action:"))
            (fexit nil)
            (w 0)
           )
           (fexit)
        (setq w (wait-for com))
        (cond   ((= w 0)    (setq fexit t))
                ((= w 1)    (transmit  "\r"))
                ((= w 2)    (transmit  "delete\r"))
        )
    )
    (transmit  "st\r")
    (wait-for '( "\rTo" ))
    (wait-for '( "Memo * Date\r" ))
    (let*   ((dels nil))
        (do*    (rec)
                ((string= (read-string 2) "M:"))
            (read-string 13)
            (setq rec (str-car (read-string 6) #\ ))
            (if (string= (read-string 1) "R")
                (setq dels (cons rec dels))
            )
            (wait-for '( "\r" ))
        )
        (do*    ()
                ((null dels))
            (transmit  "d ")
            (transmit  (car dels))
            (transmit  "\r")
            (setq dels (cdr dels))
            (mix-wait-prompt "\rM:")
        )
    )
    (transmit  "quit\r")
)

;;
;;  日経MIX用オートパイロットの本体
;;
(defun mix-auto-term ()
    (wait-for '( "\r:" ))
    (start-log)
    (mix-auto-mail)
    (mix-auto-conf)
    (if (null *up-file-list*)
        (wait-for '( "\r:" ))
        (mix-upload-sequence)
    )
    (end-log)
    (start-log)
    (transmit "who\r")
    (wait-for '( "\r:" ))
    (end-log)
)


;;
;;  日経MIXアクセス関数
;;
(defun start-mix ()
    (system "del *.ok")          ;;  古い書き込みの削除
    (gc)
    (up-list)
    (open-term *term-file*)
    (let*   ((times *max-redial*))
        (do*    ((fexit nil))
                ((or (= times 0)
                     fexit    )
                )
            (if (dialing *mix-dial-command*)
                (progn
                    (mix-login-sequence)
                    (mix-auto-term)
                    (mix-logout-sequence)
                    (setq fexit t)
                )
                (progn
                    (wait-time 100)
                    (setq times (1- times))
                    (princ "Retry !!\n")
                )
            )
        )
        (if (= times 0)
            (princ "Auto login failed !!\n")
            (princ "Auto login ended  !!\n")
        )
    )
    (end-term)
)
```


PC-9801 I/Oポートマップ
機種：2|3　ポートアドレス：0011111111011011　内部サウンド周波数設定

XLispもSchemeもそれ以上の実力をもっています.

美しい言語Lisp

何でも飲み込む括弧言語

Lisp (Common Lispから Scheme, Lisp 1.5まで含みます) はある目で見ると非常に美しい言語です. PascalやCと比べると, 括弧がたくさんあって目がチラチラして汚い……といえばたしかにそうともいえますが, 括弧はネストに対して忠実ですし, すべてS式で表現できるというのは視覚的にも感覚的にも「美しい」といってもよいように私は思います. Pascalのbegin, endのうまく使ってないプログラムを見るよりははるかに気もちのよいものです.

また, Lispの構文規則というのは, 単に「括弧がちゃんと対応していること」ということだけですから, それさえ満たしていればどんな拡張を行ってもまったくLispの雰囲気を失うことがありません. またすべての処理は関数ですから, Cなどのように「データ型は定義できるけど, 演算子の定義ができない」なんてちぐはぐなことは起きるはずもなく (元々演算子なんてないから), まったく「Lispそのもの」という趣はなくなりません. Lispの括弧はすべてを飲み込むことができるわけですね (図2参照).

同じインタプリタ言語であるBASICとの比較

インタプリタ言語としては, BASICが有名ですね. これはマクロ言語としてどうでしょうか. 最近はBASICの世界も変化してきているようなので, 昔のBASICしか知らない者が評価すると「問題発言」しかねないのですが, どうも「強力なマクロ言語」とするには, さまざまな発想の転換が必要なように思われます. というのもBASICのプログラムの基本として, 「命令」という考え方があります. ところがBASIC自身にはその「命令」を拡張する術がありません. ですから何か機能を強化しても, その機能の利用はCALLとかGOSUBとかという手続き呼び出しの命令を使ってしか行えませんから, あまり「マクロでマクロ言語自身を拡張した」という感じにできません.

また, 後の章で述べようと思っていますが, BASICのインタプリタは構文解析とかけっこうがんばらないといけません. Lispは構文がないに等しいので, 構文解析はほとんど行わなくてもよいため, その分インタプリタとしては高速に動作させることが可能なんですが, BASICだとそのへんの手間がよけいにかかります. このことは実装する手間でもいえることです. たかがマクロ言語にしてはとんでもない程手間がかかるものですし, それを楽しようとするとプログラムを書くときに, 黒魔術的技術を必要とします.

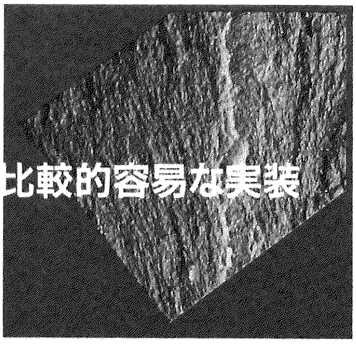
とはいえ, 「BASIC風マクロ言語」をもったソフトはけっこう多いようです. たしかにLispと比べたら, 素人目にとっつきがよいでしょうが. しかし……, 「BASIC風マクロ言語」をもったシステムをデザインする人の多くは, 「BASICぃ？　そんなの言語っていわないよぉ」なんて挑戦的発言を普段しているような人が多いはずなんですが…….

それらマジメな議論とは別に, 「この○○はぁ, マクロ言語として○○ Lispを備え……」なんてちょっとカッコイイ……と思うのはあまりにミーハーでしょうか？　別にLispはかっこ悪い言語でもないですし (括弧が多くて悪いという意見には同意しますB) これだけのメリットがあるんですから, 新たに言語をデザインするなんてむずかしいことはやめて, テキトーにLispを

図2　PL/0風Lisp拡張? (文献[1]より)

```
(program gcd
    ((const  (x = 10) (y = 45))
     (var z)
     (procedure gcd1
        ((var a b)
         (a := x)
         (b := y)
         (while  (a /= b) do
            (if  (a < b) then (b := (- b a)))
            (if  (b < a) then (a := (- a b))))
         (z := a)))
     (call gcd1)))
```

マクロ言語として採用してもちっとも悪くないと思います.



いかに強力な機能が提供できる言語仕様をもっていても，実装がむずかしい言語であれば「ちょろっと」マクロ言語として採用する……といっても容易じゃありませんね．また，実際の機能から比べてマクロ言語の部分が大きすぎるというのも何だかおかしい話です．まぁマクロ言語をもつくらいのアプリというのは，けっこう高機能なものでしょうけど.

これらの点でもLispは非常によい言語でしょう．構文が非常に単純ですから，ほかの多くの言語のように構文解析で悩むことはありませんし，tokenも単純ですから字句解析で苦労することもありません．いちばん厄介なのは記憶管理なんですが，これもマクロ言語という用途に限って使うということにしていれば，少々の手抜きをしても平気です.

ご存じの方も多いと思いますが，Lispのインタプリタは

```
(loop (print (eval (read))))
```

というごく単純なループで実現されています．つまり「S式を読む，それを評価して値を求める，その結果を印刷する」というだけの処理です．もちろんreadもprintも豪華な仕様にすればどんどん複雑になりますが，手を抜けばいくらでも簡単にできます．またマクロ言語として使うのなら結果の値にはあんまり用はないでしょうから，printそのものをまったく省くこともできるはずです．「ちょろっと」書いて数100行．「言語」といえる体裁を整えても2000行にはなりません[2]．そりゃいろいろ豪華に作れば，いくらでも大きくなりますが……，ついでにいうとPascalコンパイラの場合，字句解析だけで600行程になります[3]．数値を整数に限ってもせいぜい1，2割減る程度でしょう．もう最初から勝負になりません.

「実装するのは簡単」とはいえ，全部自分で作るのもけっこうたいへんな話ですし，ヒマも技術もない，技術はあるけどヒマがない……って方もあるでしょう．そこで「どっかからソースを入手してきてそれを使いたい」という要求が出てくると思います．幸いなことにLispはfreeな処理系が非常に多く出回っていて，それこそ「どれを使ってよいかわからない」くらいあります．個々の名前は最後の章にあげておきますが，とにかくソース付きのfreeな処理系はたくさん出回ってますから，改造の元ネタには事欠きません．自分のプログラミングの力量，組み込もうとするアプリケーションの複雑さ，盛り込みたい機能とかに合わせて選択するとよいでしょう．多くの場合，自分で作るよりよい結果をもたらすと思います．ほかの人が使っているのですから，それなりに実績のある処理系のはずです.

いずれにしても，仕様に関しては何かの処理系をそのまま実装するほうがよいでしょう．これはLispに限った話ではないとは思いますが，言語のデザインは容易じゃありませんし，1度それでプログラムが書かれてしまうと言語仕様の変更は困難なものになります．またコーディングテクニックの習得もけっこう手間のかかるものですが，既存言語そのものだとすでに習得したテクニックが使えます．またマニュアルを作るのもめんどうですし，仕様の説明にはずいぶん手間がかかりますから，たとえば「XLispと同じだよ」とか「Schemeの $R^{3.99}$ と同じだよ」といえれば，説明をするにも聞くにもずいぶん楽になります．かつてLispという言語は，

実装者の数だけ方言がある

といわれていました．Common Lispの言語仕様が意識した処理系だけでも，

- Mac Lisp
- Scheme
- Spice Lisp
- Zeta Lisp
- Standard Lisp
- NIL
- Inter Lisp

などの名前があげられています（文献[2]）．これはいわゆるメジャーな処理系で，小さいものやごくローカルな処理系はそれこそ実装者の数だけあって，同じLispでありながら全然違う[4]仕様をもっています．たかがマクロ言語に使うといっても，いまさらまったく違う言語をデザインするのは，混乱の元となるので関心しません．第一使いにくいですからね．できればCommon Lisp風かSchemeにしておいたほうがよいでしょう．どちらの仕様でも完

＊2) R^3 の仕様を満たしたScheme処理系でも，小さいのだと2000行未満で実装できます.
＊3)全体では軽く10000行を越える規模です.
＊4) BASICとFORTRANくらい違うといってもよいと思います.

全実装は容易ではないでしょうが，それでも「$+\alpha-\beta$」に留めるべきだと思います．

さてここまででおもにLispのマクロ言語としての特性の話をしてきましたが，Lispのエッセンスとは果たして何かということを考えてみたいと思います．

多くのLispの教科書には，「Lispには基本5関数というのがある」と書いてあります（文献[3]など）．これは，car, cdr, cons, atom, equalというlist処理関数のことです．これらがあればlist処理はみんな書けるということで，基本関数と呼ばれるのでしょうが，これは「たし算があればすべての演算は定義できる」というのと同じような話ですし，第一「Lispは究極のマクロ言語である」という話とはまったく関係のないことです．またLisp文化圏にない者にとっては，このあたりの話題がそもそもLispに挫折する元であったり，自分とは関係のない言語だと思ってしまう元でしょう．ですから，このへんの話はとりあえず無視してしまいましょう．必要になったときに読めばよいことです．「Lisp = list prosessor」という概念は忘れることにしましょう．

またCプログラマにとっては，Cでも書けることはCで書けばよい話ですから，普通のプログラミング言語としての機能という話もとりあえず放っておきましょう．そこで，これらを除いたところのLispの特性の話をしてきたのですが，もう一度整理すると，

- **インタプリタである**
  もちろんコンパイラもあります．
- **節操なき拡張性**
  どんなに拡張してもLispはLispである．
- **動的拡張性**
  たとえプログラムの実行中でも拡張できる．
- **自己拡張性**
  プログラム自身が自分を拡張することもできる．

というところがlist処理以外のLispの特徴でしょう．これらの特徴は

- **インタプリタである**
- **プログラムもデータもS式で表現される**

ということからきているようです．Cプログラマから見たLispの本質とは，list prosessorでもなければ関数言語でもなく，この2点ではないかと思い

COLUMN

CLOS

CLOSとは「Common Lisp Object System」の略で，Common Lispのオブジェクト指向機能に関する仕様であり，CLOSによりCommon Lispはオブジェクト指向プログラミングシステムとなる．

1984年8月にオースティンで開かれたCommon Lisp委員会の決定をうけて，同年10月にCommon Lispのオブジェクト指向機能を検討するオブジェクト指向専門委員会が活動を開始した．その中でCommon Lispのオブジェクト指向機能はCLOSの前身であるXerox社のPCL (Portable Common Loops)とSymbolics社のFlavorを軸に進展していった．その後，Daniel G. BobrowとGregor Kiczalesの両氏によってその仕様がまとめられ，1988年6月にANSI X3J13 88-002Rとして確定した．

オブジェクト指向プログラミングの重要性は，Lispに限らず，ほとんどの言語で認識されている．その基本的な考え方が理解され浸透するにしたがって，ウィンドウシステムを含むユーザーインターフェースやシステム記述，そして応用プログラミングの開発に欠くことのできないものになってきている．

Smalltalk以降に登場したオブジェクト指向言語の多くが，Smalltalkのように独立した言語ではなく，既存の言語の上にオブジェクト指向機能を付加する形で提供されたハイブリット言語であった．CLOSもこの例にもれず，Common Lispにオブジェクト指向機能を付加するものである．したがって，CLOSは基本的にCommon Lispの仕様の一部ということになる．

CLOSには，クラス定義とメソッド定義を分離している，クラス定義をせずに組み込みクラスに対して直接メソッドを定義できる，関数の一般化としてメソッドの概念を包括し，関数呼び出しの一般化としてメッセージ送信を扱う，といった多くの特徴がある．

ます．またここにCプログラマがLispも習得するメリットがあるように思います．

ここまで「Lispは究極のマクロ言語だ」と主張してきましたが，たしかにそれがすべてではありません．もっと本来のLispの特徴も考えてみたいと思います．

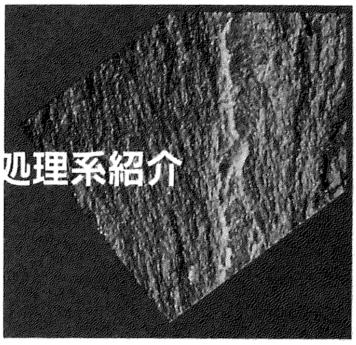
LispはFORTRANと並ぶ長い歴史をもつ言語であるにもかかわらず，長い間実用言語としてみなされず研究者やhackerのもの……つまりオタク言語として考えられていました．おかげでFORTRANのように実用言語として手垢がつくこともなく，また規格化されて硬直化することもなく育ってきました．もちろん元々の言語仕様がS式のような柔軟な構造をもったことも幸いしています．Lispは古くからあって，なおかつ今日でも新しい言語で，またいつまでも新しい……不老不死の万年青年みたいな言語です．

また柔軟でかつ関数言語としてのよい素性をもっていたため，Pascalと並んで教育用言語としても好んで使われてきました．本稿ではあえて避けてきましたが，listのような柔軟なデータ構造が簡単に使えるというのも見逃せない特徴です．何しろ実行中にデータ構造の変更ができてしまうくらい柔軟ですし，元々インタプリタ言語でしたからコンピュータ教育にはうってつけの言語です．この関係でしょうか，文献[4]のように優れたコンピュータ科学の教科書が書かれています．文献[4]は計算機の入門あたりからコンパイラの作成やハードウェアの動作までを学部の学生に教える教科書として書かれています．この本の書評は多くの本に書かれていますから，あえてあれこれ書こうとは思いませんが，プログラミングの学習に関していうなら，ほかのいろんな教科書は存在の必要すら感じません．それくらいよい本です．この本はSchemeをベースに書かれていますが，この本を読むためだけにSchemeを覚えても損はないでしょう．

文献[4]を読むといろいろと感じると思いますが，プログラマの基礎教養としてLispというのは非常によいものだなと思います．基礎教養をあまりもたずにプログラムをはじめた人は往々にして小手先のテクニックに習熟してしまい過ぎて，プログラムをもっと論理的な方向から見ることが少なくなる傾向があるようです．よく「コーディングテクニックに走るよりはアルゴリズムから見直して……」といわれますが，このあたりも基礎教養の有無がずいぶん効いてくることだと思います．

Lispという言語はよいにつけ悪いにつけ，コンピュータの物理的な存在はあまり見えてきません．またLispの実行メカニズム自体もかなり工夫されているとはいえ，Cなどから比べれば非常にトロいものです．Lispにももちろんコーディングテクニックは存在しますが，普段多くのCプログラマがやっているようなテクニックはあまり通用しません．そういった環境にあって，プログラムを書いてみるということはプログラムの論理的な側面を考える上で，大いに役に立つことだと思います．

また実際にLispでプログラムを書いてみると，世の中の多くのプログラムはCやアセンブラで書かないといけないほど高速性を要求していないこともわかるようになると思います．より高級な言語で楽に書いてもいいんじゃないかということが実感できます．Cで書くばかりが能じゃないのです．まぁ現実問題としてCほど便利に使える言語は，とくにDOS上では少ないということは認めますが．



Lispの処理系は非常に数多くありますが，今回はマクロ言語として使う話をおもに書いてきましたので，freeで流通している処理系を紹介しましょう．非常にたくさんあるのですが，私の手元にあるものだけを紹介します．

XLisp

David Betsさん作のオブジェクト指向言語として拡張された処理系です．最新はVer 2.1のようです．オブジェクト機能を除いた部分はCommon Lisp風になっています．かなり昔からある処理系ですし，移植性も高いので安心して使えます．

KCL，AKCL

京都大学数理解析研究所で作られたCommon Lispそのものの仕様をもっている処理系です．巨大な処理系ですが，最近はDOS-extenderを使ってMS-DOS上でも動くようになりました．

Franz Lisp

Franz Lispにはフリーの版と商品の版があるようです．Common Lisp以前でのメジャーな処理系の1つです．

UtiLisp

東京大学で開発された処理系です．ちょっと前のIBM系メインフレームではほとんど唯一の選択でした．最近ではSun-4とかMacとかでも動いてるようです．これもCommon Lisp以前のメジャーな処理系の1つです．

sf

sutoさん作のテキストフォーマッタですが，マクロ言語がLispです．というかemacs lispのようにテキストフォーマッタを書くために作られたLispのようです．仮想コードコンパイラももっていて，非常に高速です．

Gnu Emacs

本文の中でも説明しましたが，Free Software Foundationから出ている有名なテキストエディタですね．これも非常に巨大な処理系をもっていますが，最近DOS - extenderを使ってMS-DOS上でも動くようになりました．Lispの開発環境としても非常に使いやすいのはもちろんのこと，エディタ内蔵なので（あたりまえ）Lispの教育用としても優れています（ちなみに本稿はNEmacsで書いています）．

Schemeの一団

いろんな人がいろんな処理系を作成しています．拡張された部分を除くとだいたいR^3か$R^{3.99}$などのドキュメントに基づいて作成されていますので（最新のドキュメントはR^4です），ほとんどのプログラムで互換性があります．これからLispをはじめる人にはSchemeを（私は）お勧めします．教科書も非常によいものがあります（文献[4]，[5]）．

-UMB-Scheme

UMBで開発されたSchemeです．gccでコンパイルできます．

-MIT-Scheme

Gnu Emacsのテープに入ってます．機能は豊富なようですが，非常に大きな処理系です．

-AM-Scheme

京都大学の森脇淳さん作です．BSD，DOS，Macの上で動きます．

-jing-Scheme

同じく森脇さん作です．非常に小さい処理系で，ソースが1ファイルですが多倍長整数も使え，またけっこう高速です．

-Jafferさんの Scheme

名前が「Scheme」とだけ書かれています．Jacalという Gnuの数式処理系を動かすために作られた処理系です．非常に高速ですが，まだ多倍長整数が実装されていないようです．

-T

Yale大学で作られた処理系で，Gnu Emacsのテープに入っています．コンパイラも内蔵しているようです．

-Elk

X window用のマクロ言語とした作られた言語のようです．.Oファイルのロードができます．

-SchemeToC

DECで開発されたSchemeをCにコンパイルするコンパイラです．

XScheme

XLispと同じ作者が作っているSchemeの処理系です．またその名前から想像できると思いますが，XLisp同様にOOP拡張されています．

参考文献

[1] THEMSKY 7bits,bit Vol 19,No. 7

[2] Guy Steel Jr. Common LISP the Language,Digital Press

[3] マイコンピュータ No.15,CQ出版社

[4] H.Abelson & G.J.Sussman, Structure and Interpretation of Computer Programs,MIT Press（邦訳は，元吉文男訳,プログラムの構造と実行,マグロウヒル刊）

[5] D.P.Friedman & M.Felleisen, The Little LISPer,MIT Press（邦訳は，元吉文男／横山晶一訳,Scheme手習い,マグロウヒル刊）

特集
Cじゃなくてもいいじゃない

工藤栄太

論理プログラミングを実現する人工知能言語

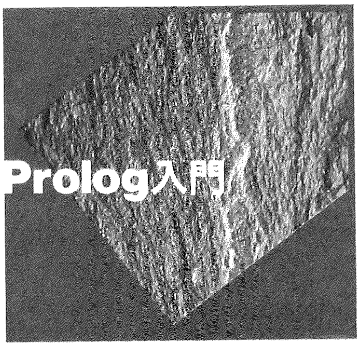
Prolog vs. C

Prologは，試行錯誤過程を組み込んだ構文解析システムとして開発され，述語論理における証明過程として説明できることからProlog（Programming in Logic）と命名された．Prologは，おもに人工知能の研究などに用いられているが，そのプログラムは人間の推論や常識をそのまま記述したかのようになっている．このPrologとCを比較するためにPrologの機能をCで実現することを考えてみる．

唐突であるが，野球とサッカーはどっちが複雑なスポーツだろうか？　多くの人はあまり躊躇なく野球だと答えるだろう．それは見た目のグラウンドの形態や，各プレーヤーの役割分担，そしてルールの数などを比べてのことだと思う．ところが１人１人のプレーヤーがプレイ中にしなければならない身体の動きはどうだろう．サッカーの選手はいつも思いもよらない格好をさせられるが，野球は比較的ある決まった動きが要求される．そしてこれはそのままルールによって規制されている分の多寡によるものと考えられる．

大胆なたとえであるが，Cがサッカーなら Prolog は野球である．Cが何でもできる万能言語であるのに対して，Prolog はルールでがんじがらめにされた中でのみある決まった動作をすることができる．

本稿の目的はCと Prolog の性能比較であるが，そのためにはまず Prolog を理解してもらわなければならない．したがって次章では，まず Prolog 入門ということからはじめたい．



論理プログラミングとは何か?

Prolog は1971年フランスのマルセイユ大学で生まれた．当時A. Colmerauer は試行錯誤過程を組み込んだ巧妙な構文解析システムを開発した．しかしこのシステムは同時に述語論理における証明過程として説明できることから，Prolog (Programming in Logic) と命名された．

Prolog にプログラミング言語としての明確な解釈を与えたのは，R. Kowalski である．彼は一階述語論理において，Horn 節と呼ばれるものに限定した場合の導出 (resolution) がこの Prolog の実行過程であると説明して，Prolog が論理型言語と呼ばれるゆえんをつくった．

Prolog が有名になったのは，それが人工知能の研究のために重要な役割を果たすと考えられてからである．すなわち，Prolog の記述そのものは，人間の推論とか常識とかいうことをそのまま表現したものになっていると考えることができる．

では，ここでいう論理的推論とはどういうことか? 本質は図１の三段論法にある．

すなわち，推論のしくみは次のように説明できる．

Aである．(事実)
AならばBである．
(A→Bという規則)
BならばCである．
(B→Cという規則)

という規則があるとき，これを逆向きに考えて，

「Cか？」

という問いがあるとする．このことを確かめるためには矢印を逆に辿って，

「Bか？」

という問いを行う．この問いに答えるには，また矢印を逆に辿って，

「Aか？」

ということを確かめにいく．この場合，「Aである」という事実が存在するため，結果としてCは確かめられたことになる．

Prolog のプログラムとは，AならばBである式のルールと，事実の集合を書き並べたものである．プログラムの実行とは，これらに対して何かの問いかけを行うことである．

Prolog を「古典的」に捉える

さて以上のような概念的説明では，実際プログラムを書く感覚がつかめな

図１

	人間であれば	過ちを犯す
ソクラテスは	人間である	
ソクラテスは		過ちを犯す

図２

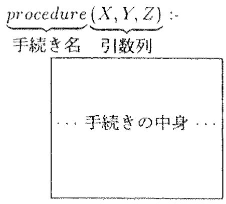


図3

```
proc(···) :- ︸ .
              空
```

リスト1
Prolog

[例]

```
create(File, Contents) :-
    open(File),
    write_out(File, Contents),
    close(File).
```

いだろう．ここでは矢印を逆向きに辿って証明云々という説明はいったんさておき，とにかく矢印の逆方向に実行が動いていくという観点のみに着目して，Prologの動作を従来型のプログラミング言語になぞらえて説明することにしよう．

手続きとはプログラム中のある動作のひとまとまりを示す．それはFORTRANやBASICではサブルーチンと呼ばれたり，Cの中では関数(function)と呼ばれたりするものである．Prologの場合も同様に，この関数がプログラムの実行単位になる(ただし次節に述べる理由で，この関数は述語と呼ばれる)．

手続き定義の中身は再びほかの手続きの呼び出しの列である(図2)．以下，例はPrologの古典的かつ代表的なシンタックスであるDEC-10 Prologに準じて表記する．

リスト1はファイルをオープンし，中身を書き出し，閉じるというよくある手続きをPrologの記法で書いたものである．定義は:-(コロン，マイナス)で，中身の手続き呼び出しの区切りはコンマで，定義の終りはピリオドで書いてある．引数のうち，変数は大文字ではじまる．子文字ではじまる名は定文字列を表す．

Prologの実行は順次書かれてある手続きを呼び出すことである．これらの手続き呼び出しは連鎖を作るが，終端の手続きとは中身が空の定義である(図3)．

いま簡単のために引数のないプログラムを記し，その実行順序を以下に示す．

```
f :- p,q.
p :- s.
q :- t,u.
s :- .
t :- v.
u :- .
v :- .
```

fという呼び出しに対しては，次の順序で実行を行う．

f → p → s → q → t → v → u

Prologの論理的意味

さてここまでの関数呼び出しの概念は普通の言語とまったく同様であるが，以上の動作を新たにProlog流に，あるいは論理プログラミング流に意味づけしておく．

一般にp(X)と書いて「Xはp(という性質)である」というふうに読む．すなわち，man(socrates)，mortal(john)，……などと書いて，「Socratesはmanである」，「Johnはmortalである」，……などと解釈できる(固有名詞は定数なので小文字ではじめる．述語の引数は一般化して複数個にしてもよい．その場合の意味づけは述語ごとに一貫していれば任意でよい)．前節までに手続き定義としてprocedure(X, Y, Z)という書き方を導入してきた．これは述語としての解釈では「X，Y，Zはprocedureという関係にある」という意味になる．Prologでは手続き定義と呼んできたものは，述語の定義として意味が付加される．だからといって，何か特別なことが必要になるわけではない．

```
p :- a,b,c.
```

は，手続きとしての実行過程と述語による推論規則としての意味づけが次のように表裏の関係にあるということである．

①手続きとして「pが呼ばれたら，その中でa，b，cを順次呼び出せ」という実行順序を述べたもの

p :- a,b,c.
⟶
実行順序

であると同時に，

②推論として，

p :- a,b,c.
⟵
推論方向

という過程を述べている．
いま，

```
p :- a,b,c.
```

という述語定義があって，

```
?- p.
```

という呼び出しが行われたとすると，これは「pか否か検証するために，a，b，cを調べにいけ」という意味になる．そして実行結果とは，この検証に成功したか失敗したかによってsuccess/failureのいずれかになる．すなわち，a，b，cに相当する事実があれば(中身が空の定義があれば)，pの証明はsuccessである．successの場合，副作用として変数にいろいろな値が情報として返ってくる．すなわち，Prologの実行過程とは呼び出しの述語の真偽を証明することである．やや乱暴な議論であるが，事実として書かれてあればそれは真である．書かれてなければそれは偽であるものと仮定する．これを閉世界仮説と呼ぶ．

例をあげると，リスト2のようなプログラムがあったときに，

```
?- mortal(plato).
```

はsuccessであり，真である．問い中のplatoはプログラム中のXと結びつけられて，man(plato)を証明にいくが，それはプログラム中に事実として存在する（図4）．

この理由で，たとえば?- man(john).は証明失敗する．これらのことを以下にもう一度まとめておく．

プログラムを書くということ
＝述語を定義すること
＝推論ルールとわかっている事実をすべて書き下すこと
プログラムを実行するということ
＝述語を呼び出すこと
＝述語の真偽を証明にいくこと

さて，次節以降はPrologだけがもつ固有の機能である．Prologの本質はパターンマッチングと後戻り制御につきる．Prologの世界では，それぞれをユニフィケーション(unification)，バックトラッキング(backtracking)と呼ぶ．これまでの説明ではPrologもほかの言語と同様，手続き逐次呼び出しとして解釈することができた．ところがPrologは，この2つの機能でほかの言語と異なる．それをこの節で説明する．

ユニフィケーション

くれぐれも誤解のないようにいうと，ユニフィケーション(単一化)あるいはパターンマッチングがProlog特有の機能であるということではない．どのような言語であってもパターンマッチングの機能は実現できる．Prologに特有なのはユニフィケーション機能がデフォルトで組み込まれているということである．

ふつうプログラミング言語の手続き呼び出しメカニズムは，手続き名の一致によった．また引数の数（arity）も一致する必要があった．Prologの場合も基本的にはこれと同様である．しかし，呼ぶ側と呼ばれる側で引数の立場は対等である．実引数，仮引数という区別はない．変数をあらかじめ宣言しておく必要もないし，そのデータタイプの宣言もない．さらに変数が現れたからといってそれがアドレスをもったある箱を表すとか，あるいはアドレスへのポインタであるとかいうこともない．まったく仮想的な箱である．このような変数は論理変数と呼ばれ，呼び出し実行メカニズムではまず，こういう考え方にもなじむ必要がある．

さて，2つの述語の呼び出し側と呼び出され側で論理変数にある定数をわりふれば一致した述語にできる場合，この2つの述語はユニファイアブル（単一化可能）であるという．?- likes(john, X)という呼び出しに対して，likes(Y, mary) :-……という述語定義があれば，X = mary, Y = johnという値のアサインができて，この2つはユニファイアブルである．ユニフィケーションはもっとずっと複雑なパターンでも可能で，f(Y, b, X)とf(a, Z, g(Z))の間では，Y = a, Z = b, X = g(b)のようにしてユニファイアブルである．繰り返しになるが，いちばん外側の述語名と引数の数は定数として一致していなければならない．

バックトラッキング

まず選択肢のある述語定義について

図4

```
mortal(plato)?
    ||
  mortal(X)      :-      man(X).
                           ||
                        man(plato)?
                      man(plato)  :-.
```

リスト2
Prolog

```
mortal(X) :- man(X).
man(socrates) :-.
man(plato) :-.
```

述べる．これは1つの述語について複数の定義がなされている場合を指す．いま述語birdの定義にはリスト3の3つがあったとする．

各々の定義の間の関係はor（まさに選択）である．したがってこれは，「Xがbirdであるならば，それは飛ぶ（fly）か，ペンギンである（penguin）か，あるいはダチョウである（ostrich）かのいずれかである」という定義になっている．どれが最初に選択されるかは純粋に書き並べた順による．

Prologの最大の特徴は，バックトラッキングである．それは証明（実行）の過程で，あるところでfailure（失敗）が起こったとすると，そこでただちに失敗を宣言して実行を終了するのではなく，ほかの選択肢のあるところまで後戻りして証明をやりなおすという機能である．

いまこの機能を上のbirdの定義で示す．同じプログラムの中にほかにリスト4の事実が書いてあったとする．

このとき，?- bird(lee).という問いかけに対して，いまいちばん最初の定義が起動されたとすると，X = leeとなって，fly(lee)を証明にいく．ところがこの世界にはfly(lee)は書かれていない．通常ならば，この時点で証明失敗で，プログラムは終了するのであるが，Prologはもっとも近い選択肢のところまで後戻りしてやりなおす．今度はbirdの2番目の定義にトライして，penguin(lee)を証明にいく．すると今度はそれに相当する事実が存在するためにこの証明はX = leeであるとして成功して終了する．

バックトラッキングは上で述べたように正解が1つ見つかるまで（あるいは，見つからないとわかるまで）走り続ける．これはときにはユーザーにとって非常に無駄な計算であってもやってしまう可能性がある．それどころか，デバッグ時にどこをどこまで戻って探し回っているのかわからないほど複雑な探索を生む必要がある．Prologではこのような事態を避けるためにcutというオペレータが用意されていて，バックトラッキングは一定範囲内でのみ起こるように制御できることになっている．ところが，このような制御オペレータは明らかに本来の論理学にセマンティクスを求めることはできない．cutオペレータは論理プログラミングの美しさを逸脱しながらもプログラミング言語としての実用性を求めた妥協点ともいうことができる．

以上，Prologの主たる特徴をまとめてきた．そのほかにも，データタイプとしてリストの扱いや複合的な項，ストリングなどの扱いに有利な仕様になっていることなどを述べる必要があるが，本稿の目的であるCとの比較の上では，とくに問題にならないのでここでは省略する．また，組み込み述語や再帰呼び出しの機能などの実行制御を述べる必要があるが，これもPrologだけのことではないのでここでは触れない．

Prologの機能をCで書いてみると……

さてPrologというのは，上で述べたように推論ということに主眼をおいた世界ではじめて真価が発揮できるような仕様になっている．したがってCとの比較で考えれば，両者には明白な役割分担があるはずで，それはどういう問題を解こうとするかによる．

いま，CがそのままでどれだけPrologの機能をフォローできるか考えてみよう．終章でも述べるが，CでPrologをインプリメントすることはもちろんできる．しかしここで述べるのは，CによるPrologの作り方というよりは，むしろそのままの言語仕様でPrologにできてCには不得手なことである．

ハノイの塔

まず，従来的によくいわれる数学の問題については両言語の差異は問題にならない．リスト5は有名な「ハノイの塔」を書き比べたものであるが，I/Oの関数，再帰呼び出しの作り方などほとんど同様であって記述量には差がな

リスト3 Prolog

```
bird(X) :- fly(X).
bird(X) :- penguin(X).
bird(X) :- ostrich(X).
```

リスト4 Prolog

```
penguin(lee) :-.
ostrich(kay) :-.
fly(nono) :-.
```


PC-9801 I/Oポートマップ
機種：④⑤⑥ ポートアドレス：XXXXXXXX100110A1 GP-IBスイッチ 使用チップ：μPD7210（7210相当）

C　リスト5

```
# include <stdio.h>
main()
{
static char l[10]="left";
static char c[10]="center";
static char r[10]="right";
int n;

printf( "Enter number >>" );
scanf("%d",&n);
move(n,c,r,l);
}

move(n,l,r,c)
int n;
char l[],c[],r[];
{
if(n == 0)
{}
else
{
move(n-1,l,c,r);
printf("move %s to %s \n", l,r);
move(n-1,c,r,l);
}
}
```

リスト5 Prolog

```
hanoi(N) :-
write("Enter number >>"),
read(N),nl,
move(N, left, right, center).


move(0,_,_,_) :-.
move(N, Left, Right, Center) :-
N1 is N-1,
move(N1, Left, Center, Right),
write("move ",Left,"to ",Right),nl,
move(N1, Center, Right, Left).
```

い．ただし，Cのほうで必要なのはデータの型を宣言しなければならないことである．Prologは最初からシンボルを扱うことを中心として考えられているのでそのへんがすでに設定されていると考えられる．

自然言語処理

さて今度は，Prologが作られるもともとの動機であった，日本語や英語などの自然言語の処理について考えてみよう．図5に英語のミニ文法を掲載する．

上記の簡約英文法は，たとえば「文(sentence)であれば，それは主語部分の名詞句（noun phrase）と述語部分の動詞句（verb phrase）からなる」，「名詞句であれば，それは冠詞(determiner)と名詞(noun)からなる」か，あるいは，「名詞と関係詞節(relative clause)からなる」などと読むことができる．最後の部分はいわゆる「辞書」に相当する．この世界では，動詞としてはwhistle, tune, ……などであると書かれている．これらは文法の静的な，データとしての記述である．ところが，Prologではこれらをこのままの形を保ってパーサ(構文解析プログラム)にすることができるのである(Warren，PereiraのDCGなど)．すなわち，図5のようなかっこうをした静的なデータを，右辺から左辺を導くような推論規則だと思うと，そのまま動的なプログラムとして稼働する．

いま簡単のため，入力文は単語の切れ目(英語文のスペースの部分)ごとに番号がわりふられているものとする．最初の文法規則は，リスト6のようなPrologのプログラムに相当する．

この中で，たとえばnoun_phrase(X，Z)は，「番号Xの位置からZの位置までは名詞句である」と読む．したがって規則全体では「入力文（XからYまで）は，適当な位置Zが存在して，XからZまでの主語名詞とZからYまでの述語動詞に分解される」と解釈される．すなわち，最初入力文があったらそれは文法規則に従って名詞句と動詞句に分解される．名詞句は冠詞と名詞に分解される．このような述語呼び出しがいくつか続いた末，最後に単語レベルを認識する辞書においてその品詞にその単語が合えばOKである．

このようにしてすべての単語がその並び順で受理されれば，その単語列は文法的な文であると判断される．どこか合わない部分が生じたら，別の規則が適用できないか，「後戻り」して再実行してみる．

たとえば関係代名詞を含んだ次のような文を考えてみよう．

“Ken whistles tunes the piano.”

人間なら，アタマから読んでいって，Kenを主語名詞だと認識し，次にwhistlesを述語動詞だと認識する．次にtunesがきたときに「あれっ」と思う．このままwhistlesを述語動詞にしておくと文が認識できない．ここでwhistlesはKenを修飾している関係代名詞だったことに気づき，やりなおす．別の規則を適用してwhistlesを関係節であるとして解析を進めるのである．するとtunesがthe pianoを伴った述語動詞であると判明して，めでた

図5

```
/* English grammar */
Sentence <- Noun_Phrase, Verb_Phrase.

Noun_Phrase <- Determiner, Noun.
Noun_Phrase <- Noun, Relative_Clause.

Verb_Phrase <- Verb.
Verb_Phrase <- Verb, Noun_Phrase.
Relative_Clause <- Verb_Phrase.

/* Lexicon */
Noun <- ken.
Noun <- piano.
Determiner <- a.
Determiner <- the.
Verb <- whistle.
Verb <- tune.

        :
```

リスト6
Prolog

```
sentence(X, Y) :- noun_phrase(X, Z), verb_phrase(Z, Y).
```

C
リスト7

```
/* 制御プログラム */

main()
{ ...
  /* 前処理など */
  ...
  apply_rule(/* 入力ストリング */);
  ...
}
apply_rule(string)
 {
  /* ルールをひとつ選択する */
  /* 入力ストリングをルールの head 部とマッチングをとる */
       if (match(string, rule_head) /* マッチングがとれれば */)
         {
           /* ルールの body 部の仕様にしたがってストリングを分割する */;
             if (/* ストリングが不可分なら */)
               {
                look_up(string) /* 辞書引き */ ;
               }
             else
             {/* 各サブストリングに対して */ apply_rule(sub_string);}
         }
        else /* 別のルールを選択する (注) */
 }

look_up(word)
 {
 }

/* 辞書 */
struct lexicon { };
static struct lexicon lex_data[] = { .... };

/* 文法規則 */
struct rule{ };
static struct rule rule_data[] = { ....  };
```

く全体は正しい文であると認識されるのである.

Prologのパーサはまさにこのような後戻り処理をともなった言語の認識を行う. この場合, noun_phraseというものの定義が2通りあって, とりあえず一方を選んで解析を進めてみたが, その先で何かまずいことが起こったので, ここまで後戻りしてきてもう一方をトライしてみたということである.

さて, この文法+辞書はそれとまったく同じ大きさのPrologのプログラムになることがわかった. このパーサをCで作るとどうなるか? 以下に推論規則, バックトラックの機能, ユニフィケーションという三点からこの問題を見てみる.

①推論規則

Prologの推論規則とて, 実行上はそれはif-then型の制御文と考えることができる. Prologでいうところの分岐のある規則群は, Cのswitch-case文であると見ればそのような書き方ができる部分もある. すなわちPrologの,

```
case(YYY) :- .... .
case(ZZZ) :- .... .
case(WWW) :- .... .
```

は, そのままCで

```
{ ...
 switch(xxx)
  {
  case yyy: ... ;
  case zzz: ... ;
  case www: ... ;
  ...
  }  ...
}
```

というふうに模倣できる. Prologのプログラミングスタイルとしてcommitted choice型(いったん選択肢を選んだら, ほかの候補にはもうバックトラックしない制御)で書けばよいということである. Prologにとってはcaseを選択することさえ普通の述語呼び出しである. ということは, 呼び出された側から呼び出した側にも値が流れるということである. Cでswitch-caseに展開してしまうと, 値の引き渡しはできない.

ただし, 以上のことはあくまで「表

面的にはそういう模倣が可能な制御もありうる」，ということである．プログラミング言語の制御構造としてif-then-elseがあるということと，プログラムで書かれた内容として推論規則があるということは根本的に違う．したがってCのプログラマという立場であれば，何か推論規則を書き下すということは，静的なデータの一種として表現するのが通常である．Prologの場合は，この静的な推論規則と動的なプログラムというものが表裏一体になっているということである．

したがってCでは文法と辞書は静的に書き下してやる必要がある．そして制御機構の分をまるまる余分にインプリメントしてやる必要がある．

リスト7のCの仕様に中で注（/* 別のルールを選択する */という部分）は要注意である．この選択機構を作るためにはバックトラックのときのこと，パターンマッチングのときのことなどすべて考慮しなければならない．このことを以下に述べる．

②バックトラッキング

本稿の目的は，バックトラック機能がない状態だと，どうひどい目にあうかということを示すことである．バックトラックがない場合は，以下の解決策がある．

1.1つには，そのような非決定的なアルゴリズムをすべて決定的（後戻りのない）アルゴリズムに書きなおすことである．そしてそのためには選択肢のあるものはすべて実行してしまって，その実行結果を保存しておく必要がある．いわゆる全解探索である．

2.Cでいっしょうけんめい戻り先と変数束縛を解くスタックを作ってちゃんと後戻り制御機構を作る．

前者はいかにも無駄な計算を生みそうである．後者はどのくらいたいへんだろうか？　まずこの機能を作ろうと思ったら，最初に思い浮かぶのは，次にどこにどう戻るべきかを記憶する必要があること，そしてそのときまでには変数バインディングのどこまでを元に戻すかということをすべて記憶しておくことが必要であることなどであろう．すなわち，戻り先の順序と制御を蓄えたスタックを作る必要があるのである．何度も繰り返しになるが，これはCのそのままの変数概念では不可能である．論理変数という仮想的な変数というものをまず実装しなければならない．筆者の知っているある処理系ではPrologの実行制御スタックは，次の4つからなり立っている．以下，規則は関数と読み替えてもかまわない．

- 実行順序記憶スタック．すなわち，1つの規則の呼び出しを1つのセルとしてスタックに積み，終了した規則のセルは解放する．
- 規則呼び出しが失敗に終った場合の，後戻り先を積んだスタック．
- ローカル変数（ある1つの規則の適用時にその規則内で束縛される変数）の束縛情報．バックトラックに備えて，その規則実行終了後でも保存しておく必要がある．
- いくつかの規則間で共用されるグローバル変数の束縛情報．やはり規則呼び出しが失敗に終わるまでスタック上のセルは解放できない．

ここまで作るならば，それはまさにCでPrologを書いているようなものである．いや，その前にもう1つのPrologの主機能を忘れていた．ユニフィケーションの機能である．

③ユニフィケーション

まずパターンマッチングの機能がPrologの第一義であるというなら，その機能をCで実現しなければならない．Cにはもちろん，論理的・数値的な意味でのイコールが組み込まれている．しかしPrologでいうユニフィケーションは，変数を含んだ表現に対して，シンボル・数値・リスト・ストリングなどを分け隔てなくあつかうイコールの概念である．そのためにはまず論理変数というものが用意されていなければならない．まずCのプログラミングに慣れた方なら，幾度かこういうような機能を作るハメになった経験があるのではないだろうか？　それは，それなりの苦労が必要である．

おわりに

何ごとにも向き不向きはある．まさかPrologを使って浮動小数点を含んだ計算をやらせようという発想はしないだろう．逆に推論規則を書いてやるのにCでパターンマッチングのメカニズムから作ろうという人はいないだろう．ただしここは注意してほしい．Cはより一般的な機能を備えた言語である．したがってCでPrologを作ることはできる．実際多くのProlog処理系がUNIXベースで動かす必要上Cで書かれている．ところが逆はありえない．あっても並たいていの苦労ではできないはずである．Prologは計算という概念のある一部だけを特化して，その部分だけを使いやすくしたものである．この点で一般的な言語Cとは役割が違うものである．

特集

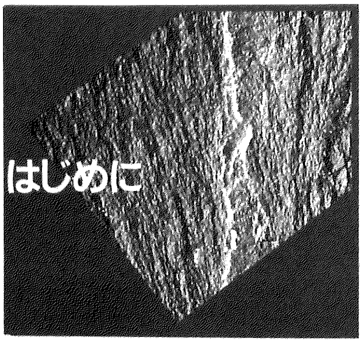
Cじゃなくてもいいじゃない

神林　靖

Smalltalk-80とインヘリタンス

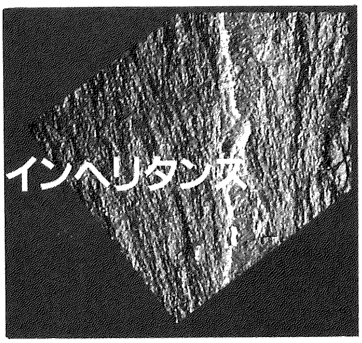
Smalltalk vs. C

C言語は，UNIXオペレーティングシステムの記述言語としてPDP-11のアセンブラ言語にとって代わるものとして設計された．それにたいしてSmalltalk-80は，未来のオフィスシステムのための創造的メディアのプログラミング言語として誕生している．この2つの対照的なプログラミング言語について，オブジェクト指向におけるインヘリタンスを通して比較し，さらにプログラミング開発環境の違いについて述べる．



Cのアンチテーゼとしての Smalltalk-80を，というのが本稿の動機である．しかしCとSmalltalk-80を比較するのは少し無理があるような気がする．衆知のようにC言語は，UNIXオペレーティングシステム開発の一環として，AT&Tベル研究所において開発された．つまりオペレーティングシステム記述言語として，PDP-11のアセンブラ言語にとって代わるものとして設計されている．それに対してSmalltalk-80は，ゼロックスの学習研究グループの研究者の手によって未来のオフィスシステムのための創造的メディアのプログラミング言語として誕生した．偶然にも時は同じく1970年代のはじめに構想されたとはいえ，かたやオペレーティングシステムの開発者が現実のシステムプログラミングの要求に応えて開発した言語であり，かたや未来を夢見る研究者が夢を実現させるために考案した言語なのだ（開発当初はSmalltalk-72，それが徐々にバージョンアップされてSmalltalk-80となった）．

この背景を考えると，同時期に開発された2つの言語がまったく異なった性格をもつ理由がよく理解できよう．誤解を恐れずに極論すれば，C言語がアセンブリ言語プラスアルファであるのに対し，なんといってもSmalltalk-80は，オブジェクト指向プログラミングの哲学をもっとも美しく実現しているプログラミング言語なのだ．したがってあらゆる面で相違を発見できるが，ここではインヘリタンスというオブジェクト指向言語に共通の機能に注目して，Smalltalk-80の紹介を行ってみたい．しかしそれだけではあまりにC言語に対して不公平なので，オブジェクト指向を実現したC言語であるC++に登場してもらい，C++におけるインヘリタンスも紹介することにしよう．



インヘリタンスとは，すでに定義されているクラスに何らかの情報（データ構造と手続き）を追加したり，変更したりすることによって新しいオブジェクトのクラスを定義することを可能にする言語機能である．オブジェクト指向プログラミングの世界では，現実世界を写し取ったさまざまなオブジェクトが動的にメッセージを交換することによって計算を行うが，このオブジェクトを系統的に表すのにインヘリタンスは強力な機構である．メッセージの送信は手続き呼び出しに類似のものであるが，後者と異なりメッセージは送られたきりで，各オブジェクトはメッセージの送受信とは別に独立に行動する．図形の描画を例にとり，メッセージの送信とインヘリタンスについて基本的な考え方を示す．

通常われわれが自分自身で図形を描くときは，四角なら四角，円なら円を描こうと思って辺の長さをそろえ，角度を直角にとり，あるいはコンパスに半径をとり，といった手順を踏むわけであるが，もしそれぞれの図形が自分自身の描画の仕組みを知っていれば，その図形各々に単に「描画せよ」と指示を与えるだけでよい．このように各オブジェクト（この場合は図形）が，自分自身のデータ構造（この場合は辺や半径）と手続き（この場合は描画の方法）をもち，外部からのメッセージ（この場合は「描画せよ」）に対応して自律的に行動するのがオブジェクト指向の基本的な考え方である．

さて，図形の世界は図1のようにそのタイプによって集合を使って分類できる．つまり図形であることを全体集

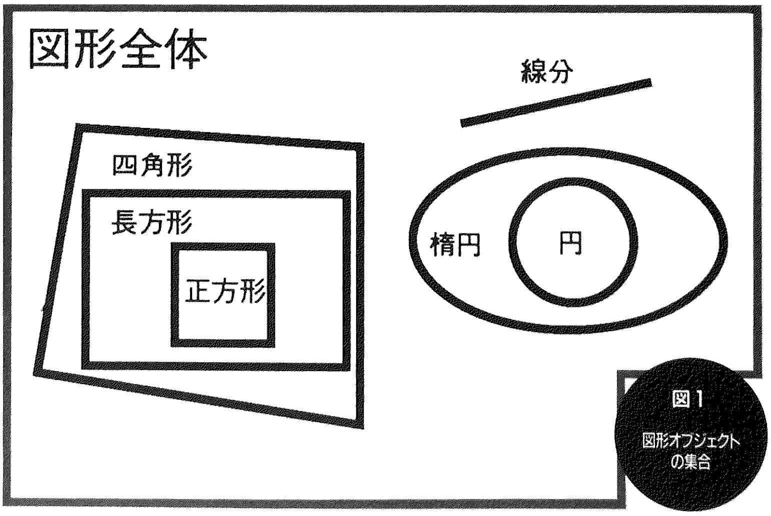


図1
図形オブジェクトの集合

合として，その部分集合として四角形をとり，またその部分集合としての長方形，正方形とより細かな分類をしていく．このとき，図の内側にあるオブジェクトはサブクラスのオブジェクトといわれ，そのオブジェクトのクラスを包含するクラスがもつ属性はすべて合わせもつ．つまり正方形は長方形でもあり，また長方形は四角形でもあるのでそれらの属性をすべてもつのである．この図のような関係をシングルインヘリタンスまたは単純継承という．

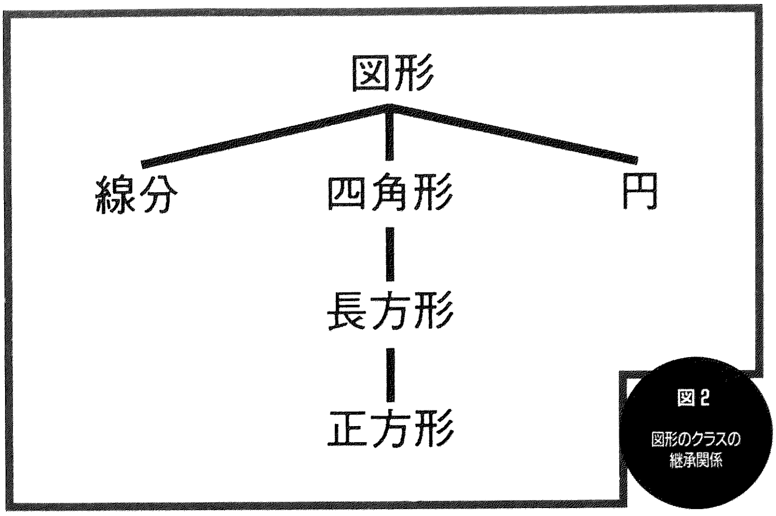


図2
図形のクラスの継承関係

継承の関係を図2に示す．もっとも四角形のような単純な場合でもシングルインヘリタンスだけではきれいに現せない場合もあり，後述するマルチプルインヘリタンス(多重継承)が必要になってくる場合もある．Smalltalk-80もC++もともに，このマルチプルインヘリタンスをサポートしている．

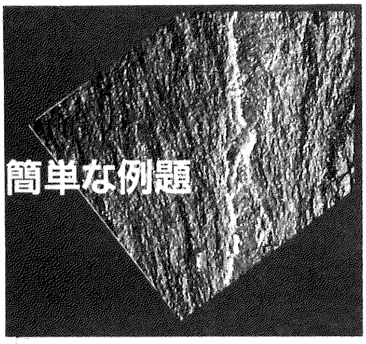


図3のような3つの図形を描くプログラムを考えてみよう．

Cに代表されるような古典的な手続き型のプログラムでは，このような図形を扱うプログラムは図形の操作をめぐって構造化されるのが通常である．操作としては，描画，回転，位置と大きさの表示といったところであろうか．図を描く手続き部はリスト1のような骨組みに沿って作成されるであろう．

ここでは図形fの形状の種類sごとにsを分類した上で，それぞれの形を描画するように定義されたコードを実行している．タイプshapeは構造体で，タグフィールドkindで形状を見分けている．

この手続き型言語でよく見られるプログラム構造の問題点は，図形を操作するプログラムコードがさまざまな手続きにまたがって存在してしまう点にある．たとえば新しい図形がこのプログラムに加えられたとすると，描画，回転，位置の表示，大きさの表示それぞれについて新たにプログラムコードを加えなければならない．しかもそのコードたるやかなりの部分が既存のコードと重複するようなものであろう．

たとえば前述の例に矢印が加えられたとする．

手続きdrawにはリスト2のようなプログラムコードが加えられる．

たとえ個々に加えられるコードの量は少なくても，似たようなコードをプログラムのあちこちに書き加えなければならない手間はかなりのものであり，エラーがないかどうかのテストはすべてやり直さなければならない．しかもエラーはどこかにこっそりと入り込んでいて，それを探すために結局プログラムの端から端までたんねんに見直さなければならないはめに陥ってしまう．

同じ問題にオブジェクト指向の考え方で取り組んでみよう．オブジェクト(ここでは図形)に自分自身のすべての

図3

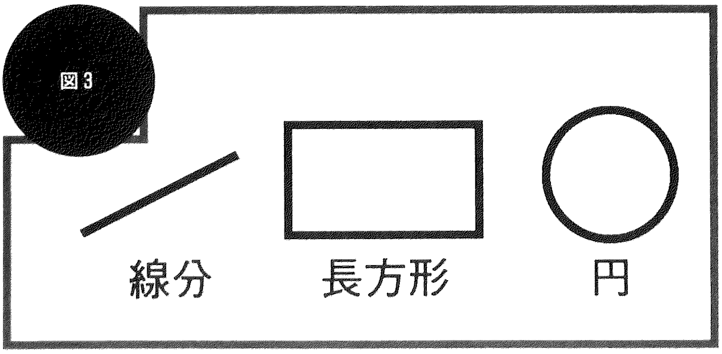
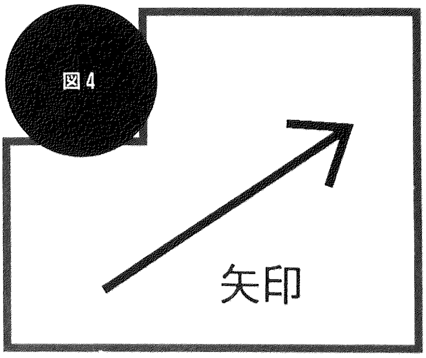


図4



C++ リスト1, 2

リスト1

```
void draw(figure f) {
        for (shape s in f) {
                switch(s.kind) {
                case LINE:      線分用のコード;
                                break;
                case RECTANGLE: 長方形用のコード;
                                break;
                case CIRCLE:    円用のコード;
                                break;
                default:        エラー処理のコード;
                }
        }
}
```

リスト2

```
void figure:: draw() {
        for( shape s in f)
                s.draw();
}
```

リスト4 Smalltalk

リスト4

```
case ARROW:     矢印用のコード;
                break;
```

C リスト3, 5

リスト3

```
void arrow:: draw() {
        line:: draw();
        draw_arrowhead();
        }
```

リスト5

```
クラス              Arrow
スーパークラス      Line
インスタンス・メソッド
draw
        super draw.             線分を描く手続きを呼び出す。
        self draw_arrowhead     矢印を描く。
```

情報をもたせるのである．たとえば図2のように図形を分類しておく．図形のクラスはすべての図形に共通な属性(たとえば位置)や操作(移動や回転)をもち，各図形に固有な属性や操作はそれぞれの図形のクラスにもたせる．

たとえば描画を考えてみると，drawという手続き名は同じでも描く手続きは円と四角形では異なる．そこでdrawとう名前でそれぞれの図形のクラス内に必要な手続きを用意しておけば，外部からはたとえ対象となる図形がなんであるかを知らなくても「drawせよ」と指示するだけでよい．

Smalltalk-80では単に

```
figure draw
```

とするだけで，メッセージdrawを受け取ったオブジェクトが，自分自身に定義されたメソッドdrawを使って自分自身を描く．C++ではリスト3のように記述することによって描画を実行させせることができる．

インヘリタンスを利用することによって，線分の部分集合として矢印を加えることも容易にできる．新しい継承関係は図5のようになる．矢印は線分のあらゆる属性を合わせもつので，描画に際して新たに加えるべき手続きは最後に矢印の頭部を書き加えることだけである．これをSmalltalk-80で表すとリスト4のようになろう．

C++では，矢印の描画に必要な手続きはリスト5のようになろう．

どちらの例でも，まず上位のクラス(スーパークラス)のメソッドdrawを使って線分を描いた後，矢印の頭部を描く自分自身の手続きを実行している．

それでは次にSmalltalk-80とC++をインヘリタンスに注目して見ていくことにしよう．

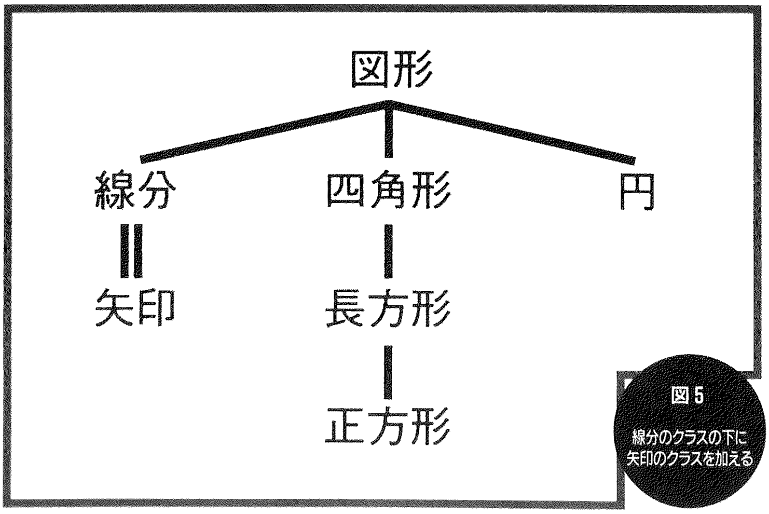
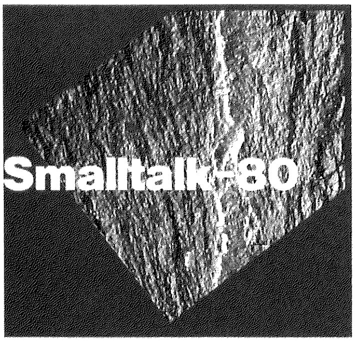


図5 線分のクラスの下に矢印のクラスを加える



Smalltalk-80には特有な言葉があるが，次の5つが基本的なものである．説明は極端に単純化してある．

①オブジェクト
データとそれを操作する手続きをひとまとめとしたもの．

②クラス
オブジェクトを集合として記述したもの．オブジェクトに共通する属性をもつ．あらゆるオブジェクトはクラスの実体である．

③インスタンス
クラスを実体化したもの．同じクラスに属するオブジェクトは共通の属性をもつが実体(インスタンス)としては異なる．

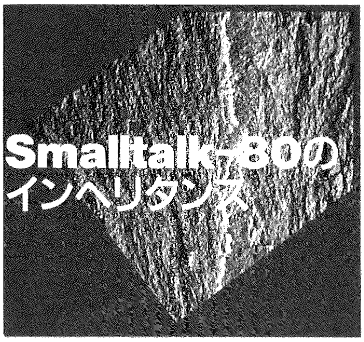
④メソッド
オブジェクトを操作する手続き．クラス内に定義され，そのクラスのオブジェクトで有効となる．

⑤メッセージ
メソッドを実行させるための要求．手続き呼び出しに相当する．

リスト6にスタックをSmalltalk-80で実現した例を，またその使用例をリスト7に示す．

この例からわかるようにSmalltalk-80は基本的に手続き型の言語である．メッセージはそれが送られるオブジェクトの右側に置かれ，適宜コロンの後ろに引数を取る．代入は←で表される．Smalltalk-80はデータ型のない言語なので，変数の定義に型宣言はなく，変数は矛盾が生じない限りどのような値でも取り得る．ここではインスタンス変数elementsとtopはメソッドinitializeが実行されたところで，それぞれ配列のクラスArrayのインスタンスと整数0となり（リスト6参照），一時変数sは，Stack newによってスタックのインスタンスとなる．しかもスタックsの第1要素は文字列Smalltalkであり，第2要素は整数80である．

このようにデータ型をなくしたことにより，柔軟なプログラミングが可能となり，プロトタイピングに適したたいへん生産性の高い言語となっている．もちろん分脈に矛盾する処理(たとえば，すでに整数が代入されている変数に文字列をさらに代入しようとした場合など)を行えばエラーとなる．コンパイル時に発見できないエラーが生じるのはこのような言語設計上やむを得ないが，実行時にエラーが起きても容易に誤りが発見できるように環境が整っているのがSmalltalk-80でもある．プログラミング環境については後述する．



Smalltalk-80ではインヘリタンスの機能を使用することにより，プログラムコードの重複を避けるだけでなく，インヘリタンスにより意味的な階層関係を表現して，プログラムを簡潔に理解しやすいものとしている．スーパークラスとサブクラスの関係は，集合とその部分集合の関係に等しい．

すべてのクラスのスーパークラスとしてObjectがある．サブクラスは，スーパークラスのもつあらゆる変数とメソッドを受け継ぐ．つまりスーパークラスのクラス変数とクラスメソッドはサブクラスでもクラス変数とクラスメソッドとなり，インスタンス変数とインスタンスメソッドもあたかもサブクラス内で定義されたようになるのである．

コンパイラを例にとり，インヘリタンスの有効性を見てみることにしよう．リスト8にコンパイラの字句解析部の一部を示す．TokenとNonterminalは，GrammarSymbolのサブクラスである．

GrammarSymbolは，Smalltalk-80システムのクラス階層の根であるObjectのサブクラスであり，オブジェクトの初期化のためのクラスメソッドnew:をもつ．インスタンス変数としてspellingを，インスタンスメソッドとしてspellとsetSpelling:の2つをもつ．GrammarSymbolのサブクラスであるTokenは，インスタンス変数spellingとGrammarSymbolのクラスおよびインスタンスメソッドを継承しているが，新たにisNullableというインスタンスメソッドも独自に加えている．同じくGrammarSymbolのサブクラスであるNonterminalは，スーパークラスのクラスメソッドであるnew:を上書きしている．

このようにサブクラスでは，スーパークラスのメソッドや変数を同名のメソッドや変数を定義することによって上書きすることができる．これによりクラス内で定義された変数やメソッドが使用される．もっともスーパークラ

PC-9801 I/O ポートマップ
機種：⑷⑸⑹　ポートアドレス：XXXXXXXX1011AAAA　RS-232C拡張インターフェース　使用チップ：μPD8251（8251相当）

ス内のメソッドも無効になったわけではないので，特別に指示することによって同名のメソッドをもつサブクラス内からも呼び出すことができる．

以下若干のSmalltalk-80の式を通して具体的に見ていくことにしよう．

Nonterminalはnew:を上書きしているが，Tokenはしていないので，「Token new: 'begin'」ではGrammarSymbolのnew:が使われるが，「Nonterminal new: 'statement'」ではNonterminalのnew:が使用される．

どちらのサブクラスもインスタンスメソッドspellは定義していないので，トークンと非終端記号のどちらのインスタンスオブジェクトxに対しても「x spell」とメッセージを送ると，GramarSymbol内のメソッドが起動されその結果に式全体が置き換わる．またどちらのサブクラスもインスタンスメソッドとしてisNullableを定義しているため，リスト9のようにすると，xはクラスTokenのインスタンスなのでTokenのisNullableが，yはNonterminalのインスタンスなのでNonterminalのisNullableが使用される．

また，リスト10のようにすると，xはクラスGrammarSymbolのインスタンスなので，isNullableというメッセージは理解できないという実行時エラーが生じる．

この簡単な例からもSmalltalk-80のインヘリタンス機能の有利さがおわかりいただけたかと思う．現実世界では事物は階層構造をしていることが多いので，それを計算機の中に写像する際にはインヘリタンスは重要となる．これによりオブジェクトの自然な分類が可能となるのでオブジェクト指向プログラミングには不可欠の要素なので

リスト6,7,8,9,10 Smalltalk

リスト6 Smalltalk-80によるスタックの実現

```
クラス             Stack
インスタンス変数   elements  top
インスタンス・メソッド
pop
        |temp|
        temp ← elements at: top.
        top ← top - 1.
        ↑temp

push: anElement
        top ← top + 1.
        elements at: top put: anElement

initialize
        elements ← Array new: 99.
        top ← 0
```

リスト7 スタックの使用例

```
|s x y|                  一時変数 s, x, y を定義する.
s ← Stack new.           s をスタックとする.
s initialize.            s にメソッド initialize を送って初期化する.
s push: 'Smalltalk'      文字列 Smalltalk を s にプッシュする.
s push: 80.              整数 80 を s にプッシュする.
x ← s pop.               x は整数 80 をもつ.
y ← s pop                y は文字列 Smalltalk をもつ.
```

リスト8 Smalltalk-80による字句解析部の一部

```
クラス            GrammarSymbol
スーパークラス    Object
インスタンス変数  spelling
クラス・メソッド
new: aString
        ...

インスタンス・メソッド
spell
        ...

setSpelling: aString
        ...

クラス            Token
スーパークラス    GrammarSymbol
インスタンス変数
インスタンス・メソッド
isNullable
        ...

クラス            Nonterminal
スーパークラス    GrammarSymbol
インスタンス変数  nullBit  productionSet
クラス・メソッド
new: aString
        ...

インスタンス・メソッド
addProduction: aProduction
        ...

isNullable
        ...

newlyNullable
        ...
```

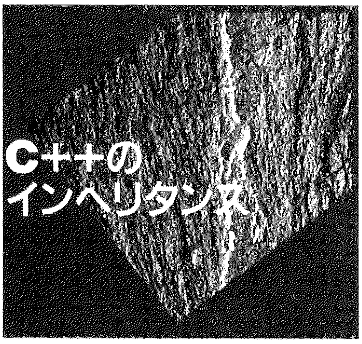
リスト9

```
        | x y |
        x ← Token new: 'begin'.
        y ← Nonterminal new: 'statement'.
        x isNullable.
        y isNullable
```

リスト10

```
        | x |
        x ← GrammarSymbol new: 'begin'.
        x isNullable.
```

ある．いうまでもないが，オブジェクト指向の言語でないCにはこのような機能はない．それではCを発展させたC＋＋ではどのようにしているか簡単に見てみよう．



C＋＋のインヘリタンス機構は，仮想関数と部分型の概念によって構成されている．ここでも例によりながらそれぞれを見ていくことにしよう．リスト11にC＋＋の単純なプログラムを示す．

クラスAにおいて，関数fは仮想関数であり，関数gはそうではない．A::fとA::gはともに文字'A'を，B::fとB::gは文字'B'をそれぞれリターンする．それでは主関数内の式b.testF()とb.testG()はどう評価されるであろうか．関数testFと関数testGは基本クラスAの中で宣言されていて，導出クラスB内でも公開されている．式b.testF()とb.testG()はそれぞれ関数fとgを呼び出すが，仮想関数の場合導出クラス内のものが優先されるので，A::testGはA::gを呼び出し'A'をリターンするが，A::testFはB::fを呼び出し，結果'B'をリターンする．

C＋＋はCと異なり強い型チェック機構があるのが優れた点であるが，それだけではインヘリタンスを実施する上で障害となる．これを解決するのが部分型の考え方である．リスト12のバッファのプログラムを見ていただきたい．

C++ リスト11, 12, 13, 14

リスト11　C++の仮想関数

```
class A {
public:
virtual char  f()        {return 'A';}
        char  g()        {return 'A';}
        char  testF()    {return f();}
        char  testG()    {return g();}
};

class B : public A {
public:
        char  f()        {return 'B';}
        char  g()        {return 'B';}
};

main() {
        B b;
        b.testF();
        b.testG();
}
```

リスト12　C++によるバッファ

```
class bufDef {
public:
virtual int     enter(char x)    { }
virtual char    leave()          { }
};

class fixBuf : public bufDef {
public:
                fixBuf() {size = MAXBUF+1; front = rear = 0;}
        int     enter(char;
        char    leave();
private:
        char    buf[MAXBUF+1];
        int     size, front, rear;
        int     succ(int i) {return(i+1) % size;}
};
```

リスト13

```
void copy(bufdef *b) {
          ...
        b->enter(c);
          ...
```

リスト14

```
main() {
        fixBuf f;
        copy(&f);
        return 0;
}
```

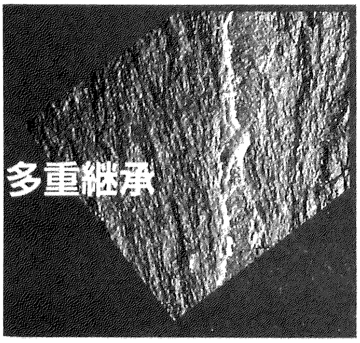
外部からはバッファを使用するに際しては，bufDefのみがわかっていればよい．たとえば関数copyを考えると，リスト13のように使用されるであろう．

fixBufはbufDefの部分型であるので，bufDefへのポインタのかわりにfixBufへのポインタを関数copyに渡すことができる．リスト14の例ではcopyはfixBufのオブジェクトであるfへのポインタを受け取っている．

こうすることによって関数copyにはなんら影響を与えることなく，バッファの実現を変更することができる．たとえば前述の例ではバッファサイズが固定されていたが，データ構造を変更してリンクリストによるバッファlinkBufをbufDefの部分型として定義したとすると，やはり関数copyはlinkBufへのポインタを受け取るであろう．

このようにC＋＋ではCでは不可能であったインヘリタンスの機構を取り入れることにより，強い型チェックと

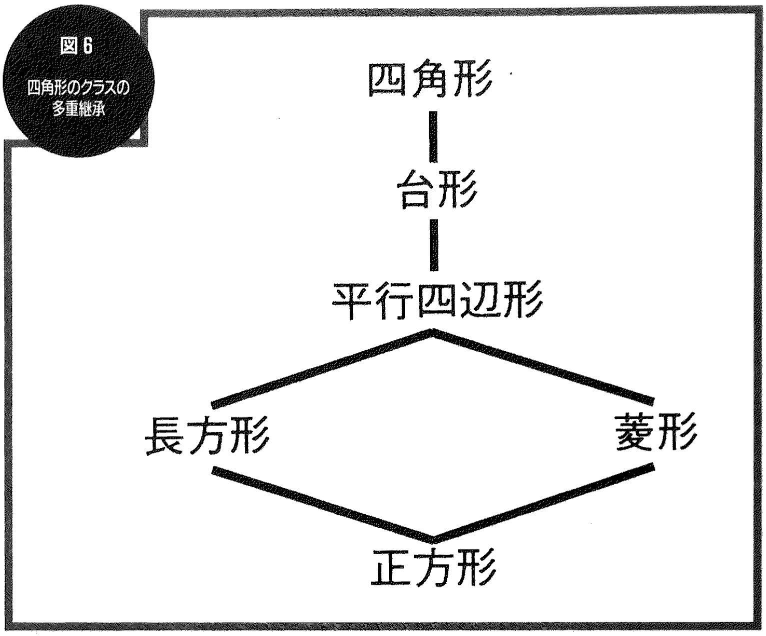
柔軟なプログラミングを同時に可能としている．Cでも柔軟なプログラミングはむろん可能であったが，それは型のチェックを緩めていたからでエラーを発生させる原因であった．それでもC++のインヘリタンス機構は後付けの感は拭えず，Smalltalk-80のすっきりとした美しさと較べると複雑でわかりにくいと思ってしまうのは筆者だけであろうか．



先にインヘリタンスの説明のところで四角形を分類したが，そのとき単純継承だけでは整理しきれないことがあると述べた．ここではそのことについて触れたい．

図6の四角形の階層関係を見ていただきたい．平行四辺形の特別な場合として長方形と菱形がある．長方形は平行四辺形に辺の長さの条件はそのままにして4隅が直角であるという属性を加えたものであり，菱形は平行四辺形の角度はそのまま4辺の長さを等しくしたものである．どちらが上位にくるようにしてもおかしく，互いに対等である．しかしここで正方形を考えると長方形，菱形の両方の属性をもつことに気づく．正方形を長方形のサブクラスとしても菱形のサブクラスとしても，角度か辺の長さかいずれかの属性が重複して定義されてしまう．ましてや平行四辺形のサブクラスにするわけにはいかない．

図6
四角形のクラスの多重継承



したがって図6のように両者をともにスーパークラスとする多重継承が実現できるとよい．これをSmalltalk-80で行うためには，Classというクラスオブジェクトにリスト15のようなメッセージを送ればよい．

リスト15のNRectangle，Rhombusはそれぞれ長方形，菱形のクラス名である．

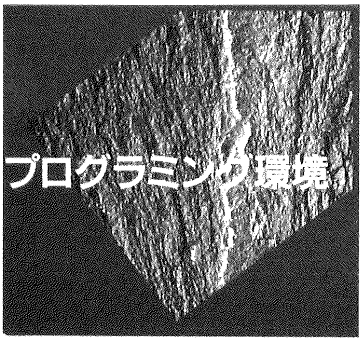
残念なことに通常のクラス定義を行う後述のシステムブラウザは単純継承のクラス階層を予定しているため，多重継承の実現のためにはこのように異なる処理を施さなくてはならない．オブジェクト指向プログラミングをもっとも美しく表している言語であるとSmalltalk-80を喧伝してきたが，それでも決して完璧なプログラミング言語ではないということである．この点C++は，プログラミング環境が整っていない昔ながらの言語なので，リスト16のようにクラス定義に際してスーパークラス名を並べればよい．いずれにしてもプログラムコードが複雑でわかりにくくなることには違いない．

リスト15
Smalltalk

```
Class named: #Square
        superclasses: 'NRectangle Rhombus'
        instanceVariableNames: ''
        classVariableNames: ''
          ...
```

C++
リスト16

```
class Square: public NRectangle, public  Rhombus {
          ...
};
```



Smalltalk-80について語る際に忘れてならないものにそのプログラミング環境がある．CやC++の開発環境といえばUNIXかMS-DOSということになるわけだが，そこにあるのは個別に存在するエディタ，リンカ，デバッガ，バージョン管理プログラムといったものたちである．本誌の読者であればviやmake, dbxといったものについて説明するまでもないであろう．

CやC++のプログラミングにmakeは欠かせないが，dbxは筆者はあまり使用しない．操作がむずかしいのだ．ソースレベルで変数の値が見れたり，トレースができるのはたいへん便利であるが，いかんせんコマンドを覚えなくてはならない．年に数回思いだしたようにCRTに向かう生活をしていると使いたいときにはすっかり忘れてしまっている．またC++では，いったんプリプロセッサがC++のソースコードをCのソースコードに落としてからコンパイルが行われるため，dbxの提供する情報とC++のソースコードとの間にギャップが生じてしまう．現在ではC++のソースコードに直接対応するデバッガも提供されているのであろうか．C++のソースコードを直接コンパイルするようにしないとむずかしいような気がするが，そうするとCを介在させることによって成立している高度な移植生が失われる危険性が生じてくる．

いずれにしてもCもC++も移植性と効率性を優先させたシステムプログラミング用の言語であるため，それほど優れたプログラミング環境が提供されているとはいえない．これに対してSmalltalk-80では事情はまったく異なる．

Smalltalk-80のプログラミング環境を簡単にまとめると次の4つが特徴となろう．

①対話型プログラミングシステム
②統合化・開放化されたプログラミング環境
③強力で洗練されたユーザーインターフェース
④体系化され豊富なプログラムライブラリ

以下，少し詳しく見ていこう．

今日ではあたりまえとなっているマルチビューシステムは，Smalltalk-80とともにゼロックス社が開発したAltoシステム上のビットマップディスプレイの上に実装された．マウスやポップアップメニューなどもこのとき開発されたものである．いわば今日われわれが享受しているプログラミング環境，計算環境のすべてが20年前に開始されたSmalltalkプロジェクトに負っているといっても過言ではない．

Smalltalk-80のプログラミングでは，原則としてすべてがシステムブラウザと呼ばれるウィンドウを通して対話形式で進められる．狭義のプログラミング（いわゆるコーディング）は，ブラウザ内のコードビューといわれるエリアに通常のテキストエディタを使って行われる．ただしブラウザは図7のようにクラスやメソッドを自然と分類しつつ編集する仕組みとなっているため，よほどへそ曲がりなことをしない限り体系的にクラスを構築しメソッドを記述していくことができる．

またこのシステムブラウザはオペレーティングシステムも含めてSmalltalk-80が提供するすべての機能をソースコードレベルで開放しているので，ユーザーはブラウザ内をいろいろと探索することによって調べるだけでなく，変更し一部を実行してみることもできる．オペレーティングシステムからコ

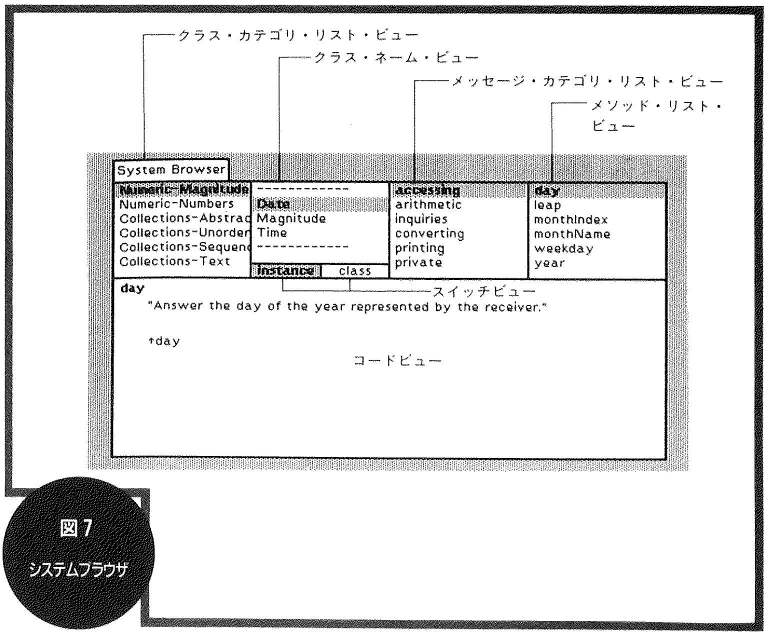


図7 システムブラウザ

PC-9801 I/O ポートマップ
機種：(1)(2)(3)　ポートアドレス：XXXXXXXX1101XXX0　未使用

ンパイラまでありとあらゆるものが(機械語で実現されているごく一部のプリミティブな部分を除いて）ユーザーに開放されているのであるから，それがどれだけ豊富なライブラリであるかは想像に難くないであろうし，ユーザーが作成するアプリケーションもオペレーティングシステムも同じように編集，実行できるのであるから，これ以上の統合化開放化は考えられない．

Smalltalk-80の環境で作業を進めていてありがたいと思うのは，この既存のプログラムコードを自由に変更して実行できる点にある．システムソフトウェアをむやみと変更するのは危険ではあるが，マウスを使って知りたいメソッドを選択し実行させることによってプログラムコードを読むだけでは知り得ない数多くの情報を学ぶことができる．むろん任意の時点で好きな変数を選択すればその内容をチェックできるし，トレースやメソッドのプロパゲーションの情報(手続き呼び出しがどのように伝搬していくか)も手に入れることができる．またコンパイルとリンクについては，システムブラウザのコードビューに書き込んだプログラムコードをアクセプト(登録)することによって自動的に行われる．

このように見ていくとSmalltalk-80の開発環境は非の打ちどころのない完成したシステムのように思えるが，あくまでSmalltalk-80という閉じた世界でのことだということを忘れてはならない．つまりSmalltalk-80というプログラミング言語を用いるためには，オペレーティングシステムまで含めたSmalltalk-80の環境が必要なのである．作成したアプリケーションだけを移植することは許されず，アプリケーションの実行もSmalltalk-80の環境下で行わなければならない．システムの細部まで開放されているかわりに，個人で責任をもたなければならないパーソナルなシステムでもある．

これらの制約のため，Smalltalk-80システムは，その完成度の高さ，美しさには誰しも賞賛の言葉を惜しまないもののエキスパートシステムや大規模システムのプロトタイピングなど限られた用途にしか使用されていない．

現在Smalltalk-80はバイトコードと呼ばれる中間言語に翻訳された上でインタプリタによって実行されているが，Smalltalk-80のソースコードをC言語に翻訳するプロジェクトが10年ほど前から各地で実施されている．Smalltalk-80にはメッセージ送信という性質上どうしても動的にリンクさせなければならない部分があり，C言語への翻訳は決して容易なことではないが，成功すれば移植性が飛躍的に高まることはいうまでもない．このようなところでもはからずしもC言語の優位性がでてしまった．決してC言語もその後継であるC++も言語自体として美しくもないし優れた言語であるとも思われないが，実際のシステムへの移植性や効率を考えた場合なくてはならない重要な言語なのである．

まとめ

すでにアプリケーションはもとより基本システムでもアセンブリ言語が使用されなくなってきている現在，C言語やC++は，もっともマシンに近い部分を受けもつシステム言語として，Smalltalk-80は，もっとも人間に近い部分を受けもつアプリケーション言語として，コンピュータサイエンスの世界の対極に位置しつつ別々の方向へ発展していくことになろう．

もっとも多くのプログラマにとってマシンがブラックボックス化していくにつれて，Cのような言語もマシンやアセンブリ言語とともに表面からは見えなくなっていくことであろうし，またそれが望ましい姿であると思う．アプリケーションには，よりドキュメンテーションに優れた言語があるのだから．

参考文献

[1] T.ケーラー，D.パターソン，神林靖訳，「はじめてのSmalltalk-80-オブジェクト指向プログラミング入門」，啓学出版，1987．

[2] 上谷晃弘，「統合化プログラミング環境-Smalltalk-80とInterlisp-D」，丸善，1987．

[3] R.Sethi，” Programming Languages: Concepts and Constructs”，Addison-Wesley Publishing，1989．

パソコン用の言語処理系

Pascal

Turbo Pascal

米Borland社開発の統合環境型Pascalコンパイラ。Ver.5.5からオブジェクト指向プログラミングを取り入れ，Ver.6.0にはクラスライブラリTurbo Visionが付属している。

製品名：Turbo Pascal 6.0
価格：20,000円
製品名：Turbo Pascal 6.0 PROFESSIONAL
価格：39,800円
問い合わせ先：ボーランド(株)
TEL 03-3486-1400

Turbo Pascal for Windows

Windowsアプリケーション開発用のPascalコンパイラ。ソースコードの編集から，コンパイル，実行までをWindows上で行うことができる。

製品名：Turbo Pascal for Windows
価格：29,800円
問い合わせ先：ボーランド(株)
TEL 03-3486-1400

Pro PASCAL

Prospero Software社のPascalコンパイラ。ISO7185スタンダードに正確に準拠し，大型機やミニコン上のPascal処理系との互換性が高い。

製品名：Pro PASCAL
価格：98,000円
問い合わせ先：(株)ライフボート
TEL 03-3293-4711

Run/PASCAL

Pro PASCALのサブセット版コンパイラ，プログラミング言語の学習環境を提供することを目的としたライフボート社のAdvanced RUNシリーズのひとつ。

製品名：Advanced RUN/PASCAL
価格：29,800円
問い合わせ先：(株)ライフボート
TEL 03-3293-4711

MODULA-2

LOGITECH MODULA-2 Ver3.0

米ロジテック社開発のMODULA-2コンパイラ。NEC PC-9800シリーズ，東芝J3100シリーズ，富士通FMRシリーズ，各社AXパソコンに対応。

製品名：コンパイラパック
価格：39,000円
製品名：ツールキット
価格：29,000円
製品名：デベロップメントシステム（コンパイラパック＋ツールキット）
価格：59,800円
製品名：Pascal to Modula-2 トランスレーター
価格：9,800円
製品名：NEW ATOM（98シリーズ用グラフィックライブラリ）
価格：19,800円
問い合わせ先：(株)NJK
TEL 03-3499-2730

TopSpeed Modula-2

IBM PCおよびその互換機用のMODULA-2処理系。Extended EditionはWindowsプログラミングに対応，OS/2 Editionは，OS/2プロテクトモードコードの生成が可能。

製品名：TopSpeed Modula-2 DOS Standard Edition
価格：199ドル
製品名：TopSpeed Modula-2 DOS Extended Edition
価格：395ドル
製品名：TopSpeed Modula-2 OS/2 Edition
価格：495ドル
問い合わせ先：Jensen & Partners International
1101 San Antonio Road Suite 301 Mountain View, CA 94043
TEL 415-967-3200

Stony Brook Modula-2

IBM PCおよびその互換機用のMODULA-2処理系。マイクロソフト社製品と互換性のあるオブジェクトの生成が可能で，1パッケージにDOS用，OS/2用，Windows用のランタイムライブラリが含まれる。

製品名：Stony Brook Professional Modula-2
価格：295ドル
問い合わせ先：Stony Brook Software
187 East Wilbur Rd,, Suite 9 Thousand Oaks, CA 91360
TEL 800-624-7487

Lisp

Waltz Lisp

Pro Code International社開発のLispインタープリタ。Frantz Lispコンパイラのサブセット。

製品名：Waltz Lisp
価格：39,800円
問い合わせ先：(株)サザンパシフィック　TEL 045-314-9514

It's CL

80386CPU用のフルセット日本語COMMON LISP.

製品名：It's CL
価格：148,000円
問い合わせ先：(株)ビー・ユー・ジー
TEL 03-3583-6890

RUN/LISP

COMMON LISP準拠のインタープリタ．ディスク上に最大8MBの仮想記憶を確保し，MS-DOS上でのAI開発を可能にしている．

製品名：Advanced RUN/LISP
価格：29,800円
問い合わせ先：(株)ライフボート
TEL 03-3293-4711

XLISP

David Michael Betzによるフリーソフトウェアの LISP処理系．現在最新のバージョンは2.1．日本では，NIFTY-Serveの人工知能フォーラム(FAI)などで手に入れることができる．

Prolog

AZ-Prolog

32ビットパソコンに対応したPrologインタプリタとコンパイラを備えたProlog処理系．

製品名：AZ-Prolog DOSエクステンダ版(グラフィックモデル)
価格：68,000円
製品名：AZ-Prolog DOSエクステンダ版(ベーシックモデル)
価格：48,000円
問い合わせ先：ソフネック(株)
TEL 0422-21-2304

Astral Prolog

グラフィックスと日本語に対応したPrologインタプリタ．Prolog-Jに上位互換性がある．

製品名：Astral Prolog(NEC PC-9801シリーズ用フル・バージョン)
価格：70,000円
製品名：Astral Prolog(NEC PC-9801シリーズ用ミニ・バージョン)
価格：6,000円
製品名：Astral Prolog(Sony NEWS用)
価格：98,000円
問い合わせ先：(有)インターメディア・アクセス TEL 03-3811-4084

Prolog-J

パソコン用として開発された，日本語対応のPrologインタプリタ．

製品名：Prolog-J(MS-DOS用)
価格：33,000円
問い合わせ先：(有)インターメディア・アクセス TEL 03-3811-4084

Arity Prolog

米Arity社開発のPrologコンパイラ&インタプリタ．DEC-10 Prolog準拠．

製品名：Arity Prolog(MS-DOS版)
価格：98,000円
製品名：Arity Prolog(OS/2版)
価格：100,000円
問い合わせ先：(株)ライフボート
TEL 03-3293-4711

RUN/PROLOG

Arity Prologとの互換性をもつ，PROLOGインタプリタ．NEC PC-9800シリーズ，東芝J-3100シリーズなどに対応．

製品名：Advanced RUN/PROLOG
価格：29,800円
問い合わせ先：(株)ライフボート
TEL 03-3293-4711

Smalltalk

Smalltalk/V

米digitalk社開発のパソコン上で動作するSmalltalkシステム．

製品名：Smalltalk/V(EPSON PC-286シリーズ，NEC PC-9800シリーズ用)
価格49,800円
製品名：Smalltalk/V DOS(IBM PCおよびその互換機用)
価格49,800円
製品名：Smalltalk/V 286(IBM PCおよびその互換機用)
価格69,800円
製品名：Smalltalk/V Mac(Macintosh用)
価格49,800円
製品名：Smalltalk/V PM(IBM PCおよびその互換機用)
価格98,000円
製品名：Smalltalk/V Windows(EPSON PC-286シリーズ，NEC PC-9800シリーズ，東芝J-3100シリーズ，IBM PCのWindows 3.0上で動作確認ずみ)
価格85,000円
問い合わせ先：(株)サザンパシフィック TEL 045-314-9514

パソコン博物館 XV

デスクトップパソコンの王様

PC-9801

日本電気PC-9801

井戸禎光

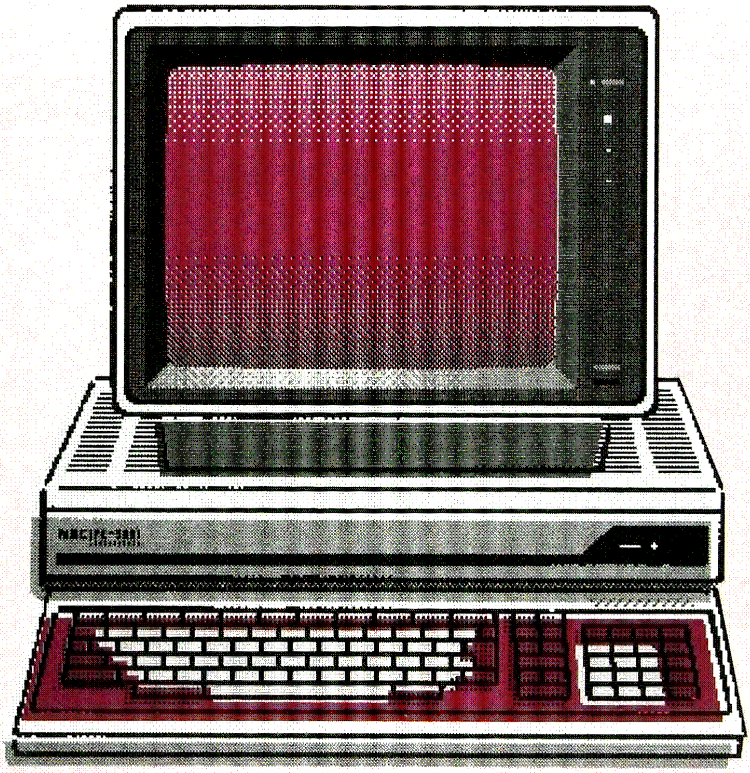


Illustration
TOSHIHITO ITO

8ビット機の世界で善戦していた日本電気が，PC-9801(1982年発売，298,000円)という16ビットのマシンを出した．価格帯の選定と当時の利用法から見たコストパフォーマンスのよさで，その後，圧倒的なシェアをとっていくのである．

しかし，世の中は未だ数年の間，完成された8ビット機の黄金時代が続いていた．幸運にも技術の流れにのっていたとはいえ，すぐに完成品ができて諸経費の回収に成功したわけではない．ただメモリ価格の低減が，より多くのメモリ容量を必要としていたアプリケーションからのニーズとあいまって，128KBから640KBへと，暫時，メモリニーズを拡大していく原動力となったことはまちがいのないところであろう．同様に，磁気記録技術，とくにフロッピーディスクの記憶容量の向上の時期と重なり，8インチから5インチインテリジェントタイプへ，5インチ2DDから2HD，3.5インチへと

目まぐるしく新製品を出すチャンスを与えられた時期であったことも確かである．

このマシンの成功の理由を探ってみたい．その第一に，価格が８ビット機の重装備タイプとあまり変わらず，速度が速く，それに何といっても16ビットは“かっこいい”ということで，初期ユーザーの心をつかんだ．第二に，いろいろな形で回路図が出版されたり，ソフトウェアの解説書が出たりして，各種の欠点をユーザーが自分で補って使うことが可能になったため，業界のいわゆるプロが使えるようになった．プロは自分の作ったソフトやI/Oを販売すべく，日夜改良を重ねていった．この時，８ビットのソフトを16ビット用に書き換える作業が割合楽であったことも１つの理由であろう．第三に，市場の反応を見て価格をあげることなく，その時期の最もニーズの多いボトルネックを解消するといった新製品の投入である．これが結果として1980年代中期の日本電気のマーケティング戦略の大成功となり，シェアがぐんぐん上昇していった．適切なニーズの吸い上げと，その設計部門へのフィードバックが，効率的に作用していたにちがいない．

一方，パソコンの利用面から見ると，８ビットデータから16ビットデータへの変更のニーズはそれほど決定的なものではなかった．しかしながら，漢字を表示するということになると，メモリの操作などで16ビットの方が簡単になる面はある．しかしアドレスが16ビットでRAMメモリの量が64KBでは非常に使いにくい（Z-80はデータ８ビット，アドレス16ビット）．アドレス面では，8086の20ビットを必要とするアプリケーションのほうがずっと多くなってきたことに対応する．別のいいかたをすれば，機械語とかアセンブラでなく，CとかFORTRANというような高級言語を使うことが多くなってきたことに対応し，それを可能にするほどメモリの価格が下がってきたことを意味する．

このようなわけで，これまでずっとPC-9800がパソコンの用途をリードしてきたように思える．実際，８ビット機のおもな用途はゲームであり，16ビット機になって一太郎とか，タイプライタ代わりのワープロといった用途が追加された．これらの用途が80%くらいといわれているが，残りには今後広がる可能性があるいろいろな用途が潜んでいる．その中で最も大きいのが表作成ソフトである．すでに企業内用途としては表作成も多い．ほかには設計の補助としてのCADとか汎用機の端末，それに最近はパソコンネットに接続し，電話を経由したコミュニケーションによる情報交換のために使われることもある．

近い将来の傾向は，“ノートブックに代表される小型化”による高機能・低価格，“32ビット，64ビットなどの多ビットCPUチップ”利用による高速化，“CD-ROMなどの光技術”利用の大記憶容量などの技術を，どのようにして取り込んでいくかの勝負であろう．

最終回

● PC-9801の外部拡張信号表

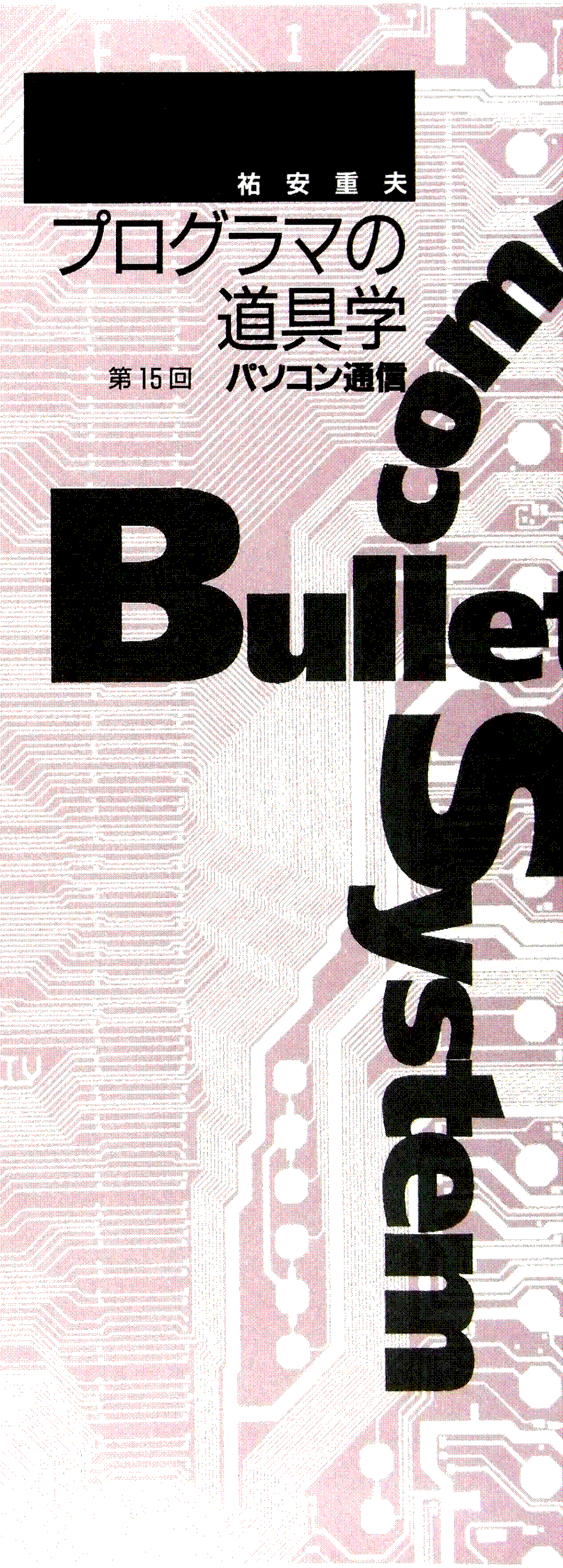
端子	信号名	I/O	注	端子	信号名	I/O	注
A1	GND			B1	GND		
A2	V1			B2	V1		
A3	V2			B3	V2		
A4	AB001	I/O		B4	DB001	I/O	
A5	AB011	I/O		B5	DB011	I/O	
A6	AB021	I/O		B6	DB021	I/O	
A7	AB031	I/O		B7	DB031	I/O	
A8	AB041	I/O		B8	DB041	I/O	
A9	AB051	I/O		B9	DB051	I/O	
A10	AB061	I/O		B10	DB061	I/O	
A11	GND			B11	GND		
A12	AB071	I/O		B12	DB071	I/O	
A13	AB081	I/O		B13	DB081	I/O	
A14	AB091	I/O		B14	DB091	I/O	
A15	AB101	I/O		B15	DB101	I/O	
A16	AB111	I/O		B16	DB111	I/O	
A17	AB121	I/O		B17	DB121	I/O	
A18	AB131	I/O		B18	DB131	I/O	
A19	AB141	I/O		B19	DB141	I/O	
A20	AB151	I/O		B20	DB151	I/O	
A21	GND			B21	GND		
A22	AB161	I/O		B22	+12V		
A23	AB171	I/O		B23	+12V		
A24	AB181	I/O		B24	IR31	I	INT0
A25	AB191	I/O		B25	IR51	I	INT1
A26	AB201	I/O		B26	IR61	I	INT2
A27	AB211	I/O		B27	IR91	I	5"HD
A28	AB221	I/O		B28	IR101	I	INT4
A29	AB231	I/O		B29	IR121	I	INT5
A30	INT0	O		B30	IR131	I	INT6
A31	GND			B31	GND		
A32	IOCHK0	I	＊1	B32	−12V		
A33	IOR0	I/O		B33	−12V		
A34	IOW0	I/O		B34	RESET0	O	
A35	MRC0	I/O		B35	DACK00	O	5"HD
A36	MWC0	I/O		B36	DACK30	O	AUX
A37	S00	I/O		B37	DRQ00	I	5"HD
A38	S10	I/O		B38	DRQ30	I	AUX
A38	S20	I/O		B39	WORD0	I	
A40	LOCK0	I/O		B40	CPKILL0	I	
A41	GND			B41	GND		
A42	CPUENB10	O		B42	RQGT0	I	＊4
A43	RFSH0	O		B43	DMATC0	O	
A44	BHE0	I/O		B44	NMI0	O	
A45	IORDY1	I		B45	MWE0	O	
A46	SCLK1	O	＊2	B46	HLDA00	O	
A47	S18CLK1	O	＊3	B47	HRQ00	I	
A48	PWR0			B48	DMAHLD0	I	
A49	+5V			B49	+5V		
A50	+5V			B50	+5V		

＊1：外部NMI
＊2：4.9152MHz
＊3：307.2KHz
＊4：バス解放要求

祐 安 重 夫

プログラマの道具学

第15回　パソコン通信



BBSなどといった略称でよばれているパソコン通信サービスに接続して，それを利用したいとおもったら，パーソナル・コンピュータとモデム，それにターミナル・エミュレータとよばれるソフトウェアを用意すればいい．あとは接続先のホストをみつけて，そこに電話をかけるだけである．

自分でパソコン通信のホスト局を開局したいとおもったら，パーソナル・コンピュータとモデム，それにホスト用のプログラムを用意すればいい，あとは誰かが電話をかけてくるのをまつだけである．

結局端末としてパソコン通信に接続するのも，ホスト局をひらくのも，ハードウェアの面ではおなじものがあればいいのである．それぞれにソフトウェアが必要だが，もちろん市販品があるし，それ以外にもフリーソフトウェアとして公開されていて，無料で入手できるものがいくつも存在する．

パソコン通信が提供するサービスは，電子メールと電子掲示板の2つにわけられる．電子メールは個人間のメッセージのやりとりであり，電子掲示板は多数にあてたメッセージである．BBSという言葉は，掲示板システム（Bulletin Board System）の頭文字をとったものである．

それ以外にフリーソフトウェアの配布などもおこなわれているが，それは電子掲示板の特殊なつかいかたにすぎない．また電子会議とよばれるものもあるが，これも構造をもった電子掲示板とかんがえればいい．

これらの機能を利用して仕事上の連絡や趣味の情報交換をおこなうのが，パソコン通信システムということになる．大手の商用サービスならば，たくさんのメンバーとの情報交換ができるし，全国的な（場合によっては全世界的な）メッセージのやりとりの道具としてつかえる．草の根BBSとよばれるものな

Illustration
Mikio Sasaki

unication n Board

ら，地域やちいさな集団の中での，きめこまかいネットワーキングのための道具として利用できる．

問題はどこのBBSに接続するか，またホスト局なら誰に接続してもらうかということの方かもしれない．道具はつかいこなしていかなければ，意味をもたない．パソコン通信も受け皿だけがあって誰もつかわないのなら，宝のもちぐされである．私もUnixワークステーションをつかって個人的なBBSを運用しているが，結局はかぎられた範囲の連絡用の道具としてつかっているだけである．

電話や郵便のようなコミュニケーションの手段は，今では私たちの日常生活からきりはなすことのできないものになっている．おなじようにパソコン通信も，すこしづつ日常の中にとりこまれていく傾向にあるようだ．

電話や郵便の利用法は，そのシステムを運営する人間が決定するのではなく，システムを破壊しないかぎりは利用する人間にまかされている．極端な話だが，犯罪の相談や脅迫に電話や郵便がつかわれたとしても，NTTや郵政省の責任になるわけではない．ましてや，それを理由に盗聴や検閲が容認されるなど，けっしてあってはならないことである．

パソコン通信はあたらしいメディアであり，個人から多数にたいしてのコミュニケーションが可能だという点では，これまでにない要素をもっている．そのため普及するにつれて，いくつかのあつれきがおこるようになってきている．今もっとも警戒すべきことは，そのことを理由に不当な法的規制がおこなわれることである．

またパソコン通信の自由でひらかれた性格になじめない自称ジャーナリストなどが，雑誌のような編集システムを導入して，無駄な(あるいは彼らが無駄だと勝手に判断した)情報を排除しようという傾向も一部にでてきているようで，これもまたこまったことである．

プログラム遊工房

谷口典生

拾四乃巻
bedit／簡易バイナリエディタ

今回は，簡単なバイナリエディタを作ってみました．ツールを作るのは，ゲームやJOKEプログラムを作るのとはまた違った楽しさがあるものです．
ワンポイントテクニックでは，プログラムの改造方法について説明しています．rawモードの使いかたなどは覚えておくと，ほかのプログラムでも参考になると思います．

リスト1

```
/*
 * bedit.c                バイナリエディタ
 */

#include <stdio.h>
#include <conio.h>
#include <signal.h>
#include <dos.h>
#include <string.h>
#include <jctype.h>

#define MAXSIZE         50000U          /* 編集できるファイルの大きさ(50KB) */
#define YBASE           3               /* 表示開始位置Y座標 */
#define YSIZE           20              /* 表示する行数 */
#define gotoxy(x,y)     printf("¥x1b[%d;%dH",y,x)

unsigned char buf[MAXSIZE];             /* ファイル内容を読み込むバッファ */
size_t filesize;                        /* 読み込んだファイルのサイズ */

void dumpl(int line, unsigned int offset)       /* 一行ダンプ表示する */
{
        int i;
        unsigned char c;

        gotoxy(1, line);
        if(offset >= filesize) {                /* ファイルの範囲外? */
                printf("¥x1b[K");               /* 行末まで消去 */
                return;
        }
        printf("%.4X  ", offset);               /* オフセットアドレス表示 */
        for(i=0; i<16; i++) {                   /* バッファ内容の16進数表示 */
                if(i>0)
                        printf(" ");
                if(offset+i < filesize)         /* ファイルの範囲内か? */
                        printf("%.2X", buf[offset+i]);
                else
                        printf("  ");           /* 範囲外は空白表示 */
        }
        printf("  ");
        for(i=0; i<16 && offset+i<filesize; i++) {      /* キャラクタ表示 */
                c = buf[offset+i];
                        if(iskanji(c) && iskanji2(buf[offset+i+1])) {
                                putch(c);               /* 漢字1文字目 */
                                putch(buf[offset+i+1]); /* 漢字2文字目 */
                                i++;
                        } else if(c<0x20 || c>0x7e)
                                putch('.');
                        else
                                putch(c);       /* 通常のASCII文字 */
        }
        printf("¥x1b[K¥n");
}

void dump(unsigned int offset)          /* 一画面分ダンプ表示する */
{
        int i;

        for(i=0; i<YSIZE; i++)
                dumpl(i+YBASE, offset+i*16);
}

void init(char *filename)               /* 初期画面作成 */
{
        int i;

        printf("¥x1b*");                /* 画面クリア */
        gotoxy(1, 1);
        printf("[%s] size=%.4XH¥n", filename, filesize);
        printf("OFFS  +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F");
        dump(0);
```

表1 コマンド一覧

[CTRL] - [E]/ [X]/ [S]/ [D]	カーソル上下左右移動
[CTRL] - [R]/ [C]	1画面スクロールdown/up
[CTRL] - [Q]/ [Z]	ファイル先頭/最後尾へ
[CTRL] - [F]/ [N]	検索文字列入力/検索実行
[CTRL] - [A]	直接ジャンプ
[ESC]	編集終了

プログラムの使い方

このプログラムはPC-9801シリーズ用で，Turbo C, QuickCおよびMS-Cに対応しています．リスト1を打ち込み，コンパイルして実行ファイルを作ってください．

プログラムの実行は

bedit 〈filename〉

と，編集したいファイル名を指定して起動してください（50KB以上の大きさのファイルは編集することができません）．

エディタを起動すると，ファイルの内容がHEX（16進数）ダンプ表示されます．左側の数字がファイル先頭からの相対位置を示し，右側には対応するキャラクタ，画面最上行には編集するファイル名とファイルサイズが表示されています．カーソルの移動はダイヤモンドカーソル（[CTRL]－[E]，[X]，[S]，[D] で上下左右）で，[CTRL]－[R]で1画面上，[CTRL]－[C]で1画面下に移動します．また，[CTRL]－[Q] でファイルの先頭へ，[CTRL]－[Z]でファイルの最後尾へジャンプします．

0～9，a～zを入力するとカーソル位置のデータを変更することができます．また，[TAB]キーによってHEX入力と文字入力のモードが切りかわり，文字入力モードでは打ち込んだ文

```
}

void inputx(int val, int addr, int pos)         /* 16進数入力の処理 */
{
        val = val<='9' ? val-'0' : val-'A'+10;
        if(pos == 0) {                          /* 上位4ビット */
                buf[addr] &= 0x0f;
                buf[addr] ^= (val & 0x0f) << 4;
        } else {                                /* 下位4ビット */
                buf[addr] &= 0xf0;
                buf[addr] ^= (val & 0x0f);
        }
}

int getkey()            /* 一文字入力 */
{
        union REGS regs;

        regs.h.ah = 0x07;               /* Direct Console Input */
        intdos(&regs, &regs);           /* DOSファンクションコール */
        return regs.h.al;
}

void edit()             /* エディタ本体 */
{
        unsigned int addr=0;            /* カーソルの指すアドレス */
        unsigned int sc_top=0;          /* 画面表示先頭アドレス */
        int pos=0;                      /* バイト内の位置(HEX入力時) */
        int mode=0;                     /* 0=HEX入力 1=文字入力 */
        int x,y;                        /* 画面内のXY位置 */
        unsigned char str[80];          /* 検索する文字列 */
        unsigned int p;                 /* 検索用変数 */
        int len;                        /* 検索文字列の長さ */
        int inkey;                      /* 入力された文字 */

        str[0]='¥0';                    /* 検索文字列を初期化 */

        do {                            /* メインループ */
                x = (addr - sc_top) % 16;
                y = (addr - sc_top) / 16;
                if(mode == 0)
                        gotoxy(x*3+7+pos, y+YBASE);     /* HEX入力モード */
                else
                        gotoxy(x+56, y+YBASE);          /* 文字入力モード */

                inkey = getkey();                       /* キー入力 */
                if(mode == 1) {                 /* 文字入力モード */
                        if(inkey >= 0x20) {     /* コントロールコードは除く */
                                buf[addr] = inkey;
                                dump1(y+YBASE, addr & 0xfff0);  /* 再表示 */
                                inkey = 'D'-'@';        /* カーソル右移動 */
                        }
                } else {                        /* HEX入力モード */
                        inkey = toupper(inkey);
                        if(inkey>='0' && inkey<='9' ||
                           inkey>='A' && inkey<='F') {  /* 16進数文字か? */
                                inputx(inkey, addr, pos);
                                dump1(y+YBASE, addr & 0xfff0);  /* 再表示 */
                                inkey = 'D'-'@';        /* カーソル右移動 */
                        }
                }
                switch(toupper(inkey)) {
                case 'E'-'@':                   /* ctrl-E カーソル上移動 */
                        if(addr >= 16)
                                addr -= 16;
                        if(addr < sc_top) {             /* 上へスクロール */
                                sc_top -= YSIZE*16;
                                dump(sc_top);
                        }
                        break;
                case 'X'-'@':                   /* ctrl-X カーソル下移動 */
                        if(addr < filesize-16)
                                addr += 16;
                        if(addr >= sc_top+YSIZE*16) {   /* 下へスクロール */
                                sc_top += YSIZE*16;
                                dump(sc_top);
                        }
                        break;
                case 'R'-'@':                   /* ctrl-R 一画面上へ */
                        if(addr == 0)
                                break;
                        if(sc_top < YSIZE*16) { /* 先頭を越えないようにする */
                                sc_top = 0;
                                addr = 0;
                        } else {
                                sc_top -= YSIZE*16;
                                addr -= YSIZE*16;
                        }
                        dump(sc_top);
                        break;
                case 'C'-'@':                   /* ctrl-C 一画面下へ */
                        if(addr == filesize-1)
```

字（コントロールコードは除く）がそのままASCIIコードとして入力されます．

[CTRL]－[F] を押すと画面最上行に「検索文字列 ＝」と表示されるので，ここで検索したい文字列を入力します．以後，[CTRL]－[N]によって入力した文字列をカーソル位置から後方に検索し，見つかった場所にジャンプします．

[CTRL]－[A] を押すとやはり最上行に「オフセット位置 ＝ 0X」と表示され，ここでHEX入力したオフセット位置に直接ジャンプすることができます．

[ESC] で編集を終了します．「書き込みますか (Y/N) ?」と聞いてくるので，編集結果を書き込んでよければ[Y]キーを，書き込まずに終了するにはそれ以外のキーを押してください．

プログラムの説明

❖dump1()

1行分のHEXダンプ表示を行います．ファイルの範囲外の部分は表示しませんが，前の画面と重ならないように空白を出力します．キャラクタ表示部分では漢字コードの2バイトを同時に出力するようにしていますが，漢字コードが行をまたぐときは行頭の表示が崩れることがあります．

```
163:                    break;
164:                if(sc_top+YSIZE*16 >= filesize) {
165:                    addr = filesize-1;
166:                } else {
167:                    sc_top += YSIZE*16;
168:                    addr += YSIZE*16;
169:                    if(addr >= filesize)
170:                        addr = filesize-1;
171:                }
172:                dump(sc_top);
173:                break;
174:            case 'Q'-'@':                   /* ctrl-Q ファイル先頭へ */
175:                if(addr == 0)
176:                    break;
177:                addr = 0;
178:                sc_top = 0;
179:                dump(sc_top);
180:                break;
181:            case 'Z'-'@':                   /* ctrl-Z ファイル最後尾へ */
182:                if(addr == filesize-1)
183:                    break;
184:                addr = filesize-1;
185:                sc_top = (addr-(YSIZE-1)*16)&0xfff0;
186:                dump(sc_top);
187:                break;
188:            case 'S'-'@':                   /* ctrl-S カーソル左移動 */
189:                if(mode == 1) {
190:                    if(addr > 0)
191:                        addr--;
192:                } else if(addr > 0) {
193:                    pos = 1-pos;
194:                    addr -= pos;
195:                } else
196:                    pos = 0;        /* ファイル先頭でストップ */
197:                if(addr < sc_top) {     /* 一行上へスクロール */
198:                    sc_top -= 0x10;
199:                    dump(sc_top);
200:                }
201:                break;
202:            case 'D'-'@':                   /* ctrl-D カーソル右移動 */
203:                if(mode == 1) {
204:                    if(addr < filesize-1)
205:                        addr++;
206:                }
207:                else if(addr < filesize-1) {
208:                    pos = 1-pos;
209:                    addr += 1-pos;
210:                } else
211:                    pos = 1;        /* ファイル最後尾でストップ */
212:                if(addr >= sc_top+YSIZE*16) {   /* 下へスクロール */
213:                    sc_top += 0x10;
214:                    dump(sc_top);
215:                }
216:                break;
217:            case 0x09:                      /* TAB 入力モード切り替え*/
218:                mode = 1-mode;
219:                break;
220:            case 'F'-'@':                   /* ctrl-F 検索文字列入力 */
221:                gotoxy(40,1);
222:                printf("検索文字列 = ¥x1b[K");
223:                scanf("%s", str);
224:                break;
225:            case 'N'-'@':                   /* ctrl-N 検索実行 */
226:                len = strlen(str);
227:                if(len == 0)
228:                    break;
229:                for(p=addr+1; p<filesize; p++) {        /* 後方検索 */
230:                    if(strncmp(&buf[p], str, len) == 0) {
231:                        addr = p;               /* 発見 */
232:                        sc_top = addr & 0xfff0;
233:                        dump(sc_top);
234:                        break;
235:                    }
236:                }
237:                break;
238:            case 'A'-'@':                   /* ctrl-A オフセット位置入力 */
239:                gotoxy(40,1);
240:                printf("オフセット位置 = 0X¥x1b[K");
241:                scanf("%x", &addr);
242:                if(addr >= filesize)
243:                    addr = filesize-1;
244:                sc_top = addr & 0xfff0;
245:                dump(sc_top);
246:                break;
247:        }
248:    } while(inkey!=0x1b);                   /* ESC で終了 */
249: }
250:
251: void readfile(char *filename)           /* ファイル読み込み */
252: {
253:     FILE *fp;
254:
```

dump()

1画面分のダンプ表示を行います．

init()

最上行にファイル名とファイルサイズ，2行目にガイドラインを表示します．

inputx()

HEX値入力のサブルーチン．入力された文字を上位／下位4ビットの値にしてバッファに書き込みます．

getkey()

1文字入力．getch()では[CTRL]－[S]や[CTRL]－[C]の入力に不都合なので，ファンクション07Hで代用します．

edit()

エディタのメインループ．

109-114行

現在アドレス（ファイル先頭からのオフセット）の画面上での相対位置を計算し，カーソルを移動する．

117-122行

文字入力モードでは，コントロールコード以外の文字コードをそのままバッファのカーソル位置に書き込みます．カーソル行を再表示し，[CTRL]－[D]が入力されたとすることでカーソル右移動の処理を行います．

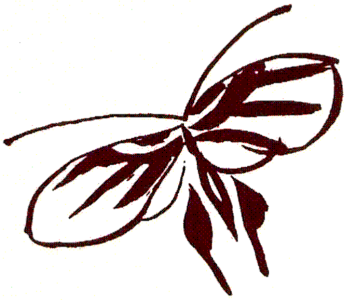
123-131行

HEXモードでは16進数文字が入力されれば値に変換してバッファに書き込み，同様の処理を行います．

132-247行

各コマンドを，switch文で分岐して処理しています．

```
        if((fp = fopen(filename, "rb")) == NULL) {
                fprintf(stderr, "ファイルがオープンできません.¥n");
                exit(0);
        }
        filesize = fread(buf, 1, MAXSIZE, fp);
        fclose(fp);
        if(filesize == MAXSIZE) {
                fprintf(stderr, "ファイルサイズが大きすぎます.¥n");
                exit(0);
        }
}

void writefile(char *filename)          /* ファイル書き込み */
{
        FILE *fp;

        if((fp = fopen(filename, "wb")) == NULL) {
                fprintf(stderr, "ファイルがオープンできません.¥n");
                exit(0);
        }
        fwrite(buf, 1, filesize, fp);
        fclose(fp);
}

void main(int ac, char **av)
{
        if(ac<2) {
                fprintf(stderr, "usage: bedit <filename>¥n");
                exit(0);
        }
        readfile(av[1]);                        /* ファイル読み込み */
        init(av[1]);                            /* 初期画面作成 */
        signal(SIGINT, SIG_IGN);                /* ctrl-C を無視 */
        edit();                                 /* エディタ本体 */
        gotoxy(1, YBASE+YSIZE);
        printf("書き込みますか(Y/N)?");
        if(toupper(getkey()) == 'Y') {
                writefile(av[1]);               /* ファイル書き込み */
                printf("¥t書き込みました");
        }
}
```



```
[bedit2.exe] size=36E9H
OFFS  +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
0280  87 52 09 2A E4 50 B8 58 00 50 E8 3B 10 83 C4 04   ⑲.*膠.X.P.:.ζ.
0290  EB C6 B8 60 00 50 E8 2F 10 83 C4 02 C7 46 FC 00    .`.P./.ζ..F..
02A0  00 EB 22 90 80 7E FE 20 72 06 80 7E FE 7E 76 06   .."逗~. r..~.~v.
02B0  B8 2E 00 EB 06 90 8A 46 FE 2A E4 50 E8 07 2A 83   .....夏F.*膠..*ζ
02C0  C4 02 FF 46 FC 83 7E FC 10 7D 47 8B 46 FC 03 46   ...F...}G祈..F
02D0  06 3B 06 50 09 73 3B 8B D8 8A 9F 52 09 88 5E FE   .;.P.s;筋粥R.  .
02E0  2A FF F6 87 2B 05 04 74 BB 8B F0 8B C3 8A 9C 53   *.  +..t.愚凝栢S
02F0  09 F6 87 2B 05 08 74 AC 50 E8 CA 29 83 C4 02 8A   .  +..t.P齪)ζ.割
0300  84 53 09 2A E4 50 E8 BD 29 83 C4 02 FF 46 FC EB   「.*膠霞)ζ..F
0310  B1 90 B8 63 00 50 E8 AF 0F 83 C4 02 5E 8B E5 5D   .精c.P隷.ζ.^句]
0320  C3 90 55 8B EC B8 02 00 E8 3D 09 C7 46 FE 00 00   .振駆....=..F...
0330  B1 04 8B 46 FE D3 E0 03 46 04 50 8B 46 FE 05 03   ..祈....F.P祈...
0340  00 50 E8 CB FE 83 C4 04 FF 46 FE 83 7E FE 14 7C   .P霊.ζ..F.ミ..|
0350  DF 8B E5 5D C3 90 55 8B EC B8 02 00 E8 09 09 B8   .句].振駆.......
0360  68 00 50 E8 62 0F 83 C4 02 B8 01 00 50 50 B8 6B   h.P璃.ζ....PP.k
0370  00 50 E8 53 0F 83 C4 06 FF 36 50 09 FF 76 04 B8   .P璽.ζ..6P..v..
0380  74 00 50 E8 42 0F 83 C4 06 B8 85 00 50 E8 38 0F   t.P錚.ζ....P.8.
0390  83 C4 02 2B C0 50 E8 89 FF 8B E5 5D C3 90 55 8B   ζ.+.P閤.句].振駆
03A0  EC 33 C0 E8 C2 08 83 7E 04 39 7E 06 B8 37 00 EB   .3.嚢.ミ.9~..7..
03B0  04 90 B8 30 00 29 46 04 83 7E 08 00 75 12 8B 5E   .精0.)F.ミ..u.疑

 C1   CU   CA   S1   SU    VOID  NWL  INS  REP  ^Z
```

readfile()

ファイル内容をバッファへ読み込みます．ファイルがオープンできなかったり，ファイルサイズが大きすぎる場合はエラー表示をして終了します（ここで文字列をstderrに出力しているのは，標準出力がファイルなどにリダイレクトされている場合でもエラーメッセージを画面に出すためです．このプログラムに関しては出力をリダイレクトして使用することはまずありませんが，フィルタとして使うようなプログラムでは重要です）．

writefile()

ファイルの書き込みを行います．もしバックアップファイルを作るようにしたければ，ここでファイルをオープンする前に元のファイルを「.BAK」などにリネームすればよいでしょう．

ワンポイントテクニック

bedit.cはバイナリエディタとして最小限の機能しかもっていません．自分なりに改造して，使いやすくしてみてください．大きいファイルを扱えるようにするのはめんどうかもしれませんが，ちょっとした機能の追加や操作を好みに合わせたりすることならすぐにできるでしょう．

たとえばstricmp () のかわりにstrnicmp () を使えば大文字小文字の区別をしないサーチになります（ただ

し漢字コードには適さない)．16進数によるサーチがあっても便利でしょうし，文字列の入力にはscanfを使うより自前のライン入力関数を作ったほうがよいでしょう．画面出力をテキストVRAMに直接書くようにすればスクロールも速くなり，ずっと快適になります．

以下に2つの改造例を示します（付録ディスク中のbedit2.cはbedit.cに以下のルーチンを付け加えたものです)．ここで使っているキーコードの設定やrawモードの使いかたは，覚えておくと結構役にたつものです．

カーソルキーを使う

カーソルキー（矢印キー）などの特殊キーには，DOSのKEYコマンドなどでキーコードを自由に割り当てることができます．矢印キーをカーソルの移動に使うため，DOSの拡張ファンクションを使い，プログラムの中で特定のキーコードを設定してやることにします．

リスト2はsetkey_list [] で指定したキーコードの設定と復帰を行うものです．[RollUp]/ [RollDown] に[CTRL]－[R]/ [C] を，矢印キーに[CTRL]－[E] [X] [S] [D] を，[HOMECLR]/ [SHIFT]＋[HOMECLR]に[CTRL]－[Q]/[Z]をそれぞれ割り当てます．

rawモードを使う

通常，printfなどで文字表示を行っているとき [CTRL]－[C] が押されると，「^C」と改行コードが画面に出てしまいます．beditでは1画面下へのスクロールを [CTRL]－[C] で行うため，[CTRL]－[C]を押しっぱなしにしていると画面が乱れてしまいます．これを防ぐには，標準出力のキャラクタデバイスをrawモードに設定してやる必要があります（通常のモードはcookedモードという)．

リスト3がこれを行うプログラムで，main関数のedit () を呼び出す前後でそれぞれrawモード，cookedモードを設定してやればよいのです．もちろん，同時に [CTRL] － [C] を無視するようにする（リスト1の287行目）のを忘れてはいけません．

リスト2

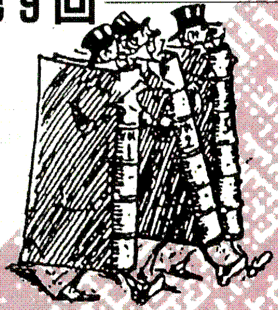
```
/* リスト2 */

char keysave[8][6];
char keyset[6] = { 0, 0, 0, 0, 0, 0 };
char setkey_list[8][2] = { { 0x15, 'C'-'@' }, { 0x16, 'R'-'@' },
          { 0x19, 'E'-'@' }, { 0x1a, 'S'-'@' }, { 0x1b, 'D'-'@' },
          { 0x1c, 'X'-'@' }, { 0x1d, 'Q'-'@' }, { 0x1f, 'Z'-'@' } };

void set_cursorkey()
{
        union REGS regs;
        int i;

        for(i=0;i<8;i++) {
                regs.h.cl = 0x0c;                       /* キーの取得 */
                regs.x.dx = (int)keysave[i];
                regs.x.ax = setkey_list[i][0];
                int86( 0xdc, &regs, &regs);             /* 拡張ファンクションコール */
                keyset[0] = setkey_list[i][1];
                regs.h.cl = 0x0d;                       /* キーの設定 */
                regs.x.dx = (int)keyset;
                regs.x.ax = setkey_list[i][0];
                int86( 0xdc, &regs, &regs);
        }
}

void restore_cursorkey()
{
        union REGS regs;
        int i;

        for(i=0;i<8;i++) {
                regs.h.cl = 0x0d;                       /* キーの設定 */
                regs.x.dx = (int)keysave[i];
                regs.x.ax = setkey_list[i][0];
                int86( 0xdc, &regs, &regs);
        }
}
```

リスト3

```
/* リスト3 */

void setmode(int mode)              /* 0:raw mode  1:cooked mode */
{
        union REGS regs;

        regs.x.ax = 0x4400;                 /* Get IOCTL Data */
        regs.x.bx = 1;                      /* STDOUT */
        intdos(&regs, &regs);
        regs.h.dh = 0;
        if(mode==0)
                regs.h.dl |= 0x20;          /* set bit5 */
        else
                regs.h.dl &= ~0x20;         /* clear bit5 */
        regs.x.ax = 0x4401;                 /* Set IOCTL Data */
        regs.x.bx = 1;                      /* STDOUT */
        intdos(&regs, &regs);
}
```

IC千夜一夜

第9回



ビット・スライス型マイクロプロセッサとマイクロプログラム ——その2——

虎岩 登

マイクロプログラム方式

その原理

では，今回はビット・スライス型マイクロプロセッサの命令解釈機構によく使われていた，マイクロプログラム方式という制御方式について説明します．

マイクロプログラム方式とは，簡単にいってしまえば，コンピュータの機械命令を解釈実行する際に，それをハードウェアに密着した特別なプログラムで書かれたインタプリタで実行する方式をいいます．このとき，そのハードウェアに密着した特別なプログラムをマイクロプログラムと呼びます．

BASICという高級言語を機械命令として直接実行できる計算機は一般的には見かけませんし，無理に製作しても複雑で高価なものになると思われます．しかし，BASICを解釈するプログラムを機械命令を使って書き上げてそれをメモリ内に入れてしまえば，プログラマの目には，あたかもBASICを解釈するコンピュータのように見えます．そうすれば，機械命令のレベルでは単純な命令だけを実装すればよいのですから，ハードウェアを単純なものにすることが可能になります．

同じことをCPU内部で行ってCPUのハードウェアを単純にしようというのが，マイクロプログラム方式なのですが，通常のインタプリタと比較すると異なる点がいくつかあります．1つは，命令の実行速度です．機械命令を解釈するための仕組みですから，ここが遅ければCPUの命令実行速度も遅くなり実用的ではなくなります．次に，実行するハードウェアの規模です．そもそもCPUの機械命令を単純なハードウェアで解釈実行するための仕組みなのですから，マイクロプログラムの解釈の際に複雑な回路を用いたのでは何のためなのかわからなくなります．

それに付随して命令の大きさの制約が違います．機械命令では，メモリ効率や実行速度の点で有利になるので，命令長は小さいほうが好まれます．同じ機能の命令セットなら，すべて4Byteで表現される命令セットより，一部でも1Byte命令を含む命令セットの方が，プログラムサイズを小さくできますし，メモリから命令を読みだすときの時間が節約されて高速になる可能性があります．ところが，命令長を短くしようとすると，解釈を行う回路が複雑になります．

マイクロプログラムでは，実行速度を重視しますので，固定長の命令を用

いて，しかも命令内部のフィールドは特定の用途にしか使わないようにするのが一般的です．したがって，1命令が数十bitからなることは普通で，大きなものだと100bitを越える場合もあります．マイクロプログラムは通常のプログラムと違って，主記憶より1桁くらい高速ではありますが，数百Wordからせいぜい数十KWordの大きさのメモリに格納されます．ですから，メモリ効率にとくに注意をはらう必要もありません．

CPU設計に見る マイクロプログラム制御

抽象的な説明をしてもわかりにくいでしょうから，簡単なCPUを大雑把に設計してしまいましょう．用意されている道具立ては図1のようなものです．レジスタが4つあって，ALUが8種類の演算を行えます．また，ALUの2つの入力にはバッファレジスタがそれぞれついており，X1とX2と名付けられています．出力にはバッファ（一種のスイッチ）があります．とりあえず，これだけの回路の中で，マイクロプログラム制御の例を見ていきましょう．

それぞれの回路はデータをやりとりするバスで接続されています．さらに全体のタイミングをとるためのクロック信号がすべての回路に行き渡っています．タイミングを必要とする回路では，この信号が0から1へ変化する瞬間（これを信号が立ち上がると呼んだりします）に何かが起こるように決められています．それ以外に回路ブロックごとに1ないし数本の信号線があります．まず，その役割を調べておきましょう．

レジスタには4本の信号線があります．そのうちの2本はアドレス線で，表1のような組み合わせでレジスタを選ぶのに使われます．1本はRRと名付けられていて，この信号が1になったらアドレス線で指定されたレジスタからデータが出力されます．最後の1本がRWと名付けられた信号で，この信号が1の場合，クロック信号の立ち上がりの瞬間にアドレス線で指定されたレジスタへデータが書き込まれます．

ALUには演算の種類を指定する信号が3本あります．その対応関係を表2に示します．出力信号としてキャリーフラグに相当するC信号があります．さらに入力のバッファレジスタにはそれぞれ1本ずつの信号線があります．それぞれW1，W2と名付けてあります．この信号が1だと，クロック信

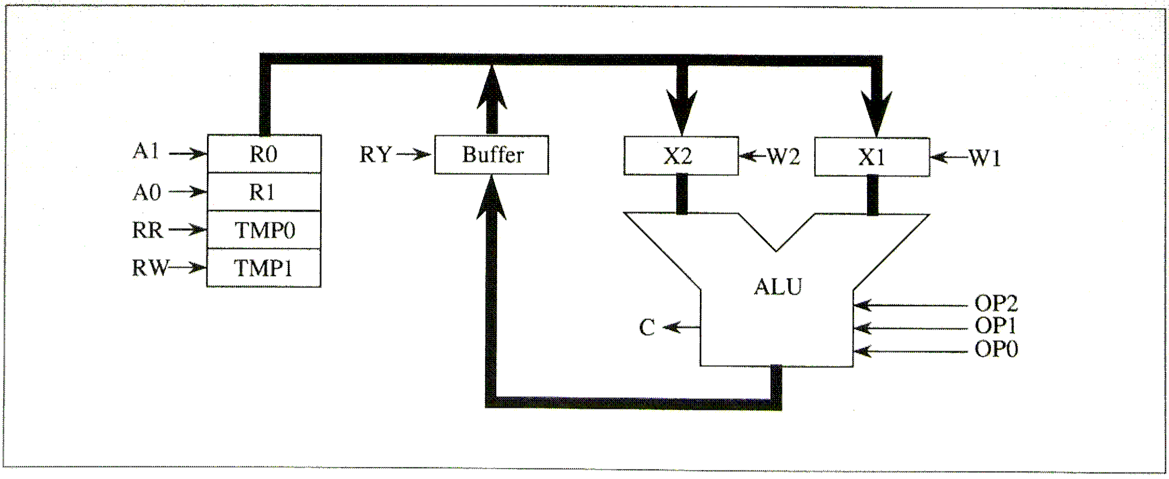
表1

A1	A0	対応レジスタ
0	0	R0
0	1	R1
0	0	TMP0
0	1	TMP1

表2

Op2	Op1	Op0	演算
0	0	0y = X1	
0	0	1	y = x1 & x2
0	1	0	y = x1 | x2
0	1	1	y = ~x1
1	0	0	y = x1 + x2
1	0	1	y = x1 − x2
1	1	0	y = x1 >> 1
1	1	1	y = x1 << 1

図1　仮想CPUの内部ブロック図



号の立ち上がりでデータバス上のデータが書き込まれます．ALUの出力バッファにはRYという信号があり，この信号が1だとALUの出力がデータバス上に出力されます．

演算，レジスタ関係の回路は以上のようなものです．あとは，これらの信号を適切なタイミングで変化させてやれば，さまざまな演算が可能になります．しかし，複雑な命令を簡単な回路で実現しようとすれば，これら制御信号を変化するところが複雑になりすぎはしないでしょうか．

そこで，マイクロプログラムという手段で，複雑さを整理してしまおうという考え方が登場します．

この場合，図2のようなマイクロ命令のフォーマットを作ってしまいます．単に制御信号を並べただけです．ただ，一番先頭に，マイクロ命令の実行が終わることを知らせるビットが付けられています．そうして，マイクロ命令を特別なROMに入れておいて，あとは，クロック信号が立ち上がるごとに順番に読みだしてやります．そうすれば，何十ステップもの複雑なタイミング制御が，たいして複雑な回路を用いずに実現できるわけです．つまり，一種のプログラム，マイクロプログラムができたのです．メモリというのは，他のロジック回路より集積度が上げやすいですし，マイクロプログラムのサイズは小さいのが普通ですから，特別に高速なメモリを使っても，それほどコストアップにはつながりません．

では，実際にプログラムしてみましょう．

まずはレジスタ間の加算から．

```
ADD    R0,R1 (R0 = R0+R1)
```

この演算は，

図2　マイクロ命令のフォーマット

E	A1	A0	RR	RW	Op2	Op1	Op0	W2	W1	RY

Eは最後の命令を表すE(nd) フィールド

表3

E	A1	A0	RR	RW	Op2	Op1	Op0	W2	W1	RY
0	0	0	1	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	1	0	0
1	0	0	0	1	1	0	0	0	0	1

```
X1 = R0
X2 = R1
R0 = R0 + R1
```

という風に分解して，順番に実行すればよいので，マイクロプログラムは表3のような命令列で表されます．

また，説明していませんでしたが，マイクロプログラムは，CPUが機械命令を読み出したときに実行が開始されます．マイクロプログラムにも，プログラムカウンタが存在しています．CPUが命令を読み出したら，マイクロプログラムのどのアドレスから実行するのかを，一種の表を引いて機械命令から求めます．表はROMを使えば実現できます．そうして，マイクロプログラムの先頭ビットが1になったら，次の機械命令を読みだすことにします．こうすれば，次々と機械命令を解釈していく回路，つまりCPUが完成するわけですね．

このマイクロプログラムのプログラム実行を制御する部分を，シーケンサと呼びます．これまで考えてきたCPU向けのシーケンサの構成を図3に示します．＋1と書いてあるのはインクリメンタと呼ばれる回路で，入力に1を加えて出力するものです．ついでに，マイクロプログラムで条件分岐ができるようにマイクロプログラムを拡張しておきました．図4のようにフォーマットを変えます．変えるといっても，JCビットと10bitのジャンプアドレスのフィールドを追加しただけです．マイクロプログラムは1KWordの大きさと仮定して，10bitで指定することにしています．JCビットが1で，かつキャリーフラグに相当するC信号が1のときに，図3の切り替えスイッチを操作して，ジャンプアドレスフィールドの値をマイクロプログラムカウンタにロードしてしまいます．これが条件分岐の方法です．

条件分岐が可能になったので，複雑な命令が実現できます．たとえば乗算を実現してみましょう．乗算には，筆算と同じ，1桁ずつずらしながら足していくアルゴリズムを使いましょう．では，

```
MPY    R0, R1 (R0 = R0*R1)
```

を，次のように分解してみます．

```
TMP0 = R0
TMP1 = 0
X2 = R1
```

```
TMP0 = TMP0 << 1
if (carry==1) {
    TMP1 = TMP1 + X2
}
TMP1 = TMP1 << 1

TMP0 = TMP0 << 1
if (carry==1) {
    TMP1 = TMP1 + X2
}
TMP1 = TMP1 << 1
    ⋮
```

(レジスタのビット数分繰り返す)

だいたいの概要がわかれば，各ステップは1ないし数命令のマイクロ命令にさらに分解できます．たとえば，最初の

```
TMP0 = R0
```

は，表4の2ステップです．次の

```
TMP1 = 0
```

は，ちょっとテクニックが必要で，常にx - x == 0であることを利用して，表5のようにします．ALUの2つの入力レジスタに同時にR0をロードしていますが，こういう使い方もハードウェアの構成によってはできるのです．

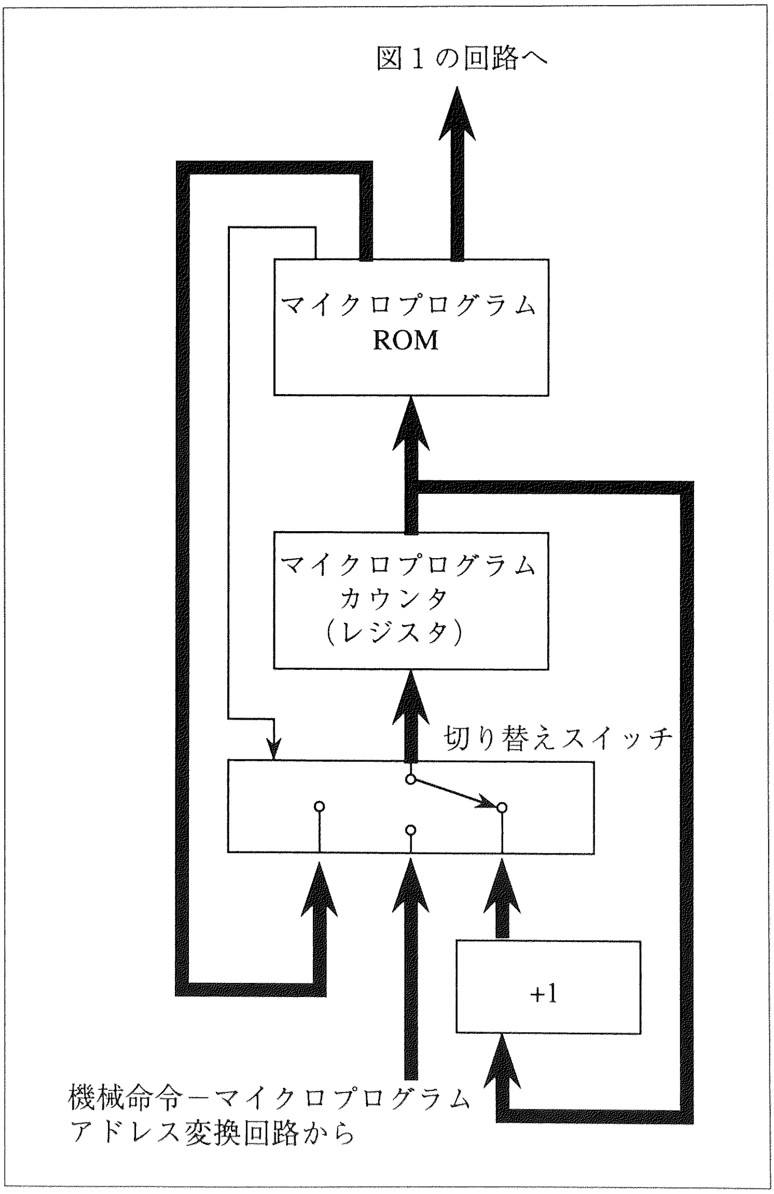
あとは，皆さんでコーディングしてみてください．

これまでに設計したCPUの中には，重要なものが省略されています．外部との入出力の部分です．特殊なレジスタに書き込むと，それがCPUの外側のアドレスバスに出力されたり，その数クロック後にはメモリから読みだされたデータがもう1つべつの特殊なレジスタから読み出すことができるようになっているというような機構を付ける必要があります．また，命令の一部を読み出すことができるような特殊レジスタも付けておくと命令解釈が簡単になります．

そうすれば，インデックスレジスタを使ったインデックスアドレッシングの計算などを命令読み出し後に行って，さらにそのデータに演算をするような命令も実現できます．アドレッシング計算などは，どの命令でも必要になることが多いですから，ソフトウェアから見ればサブルーチンにするのが適当な気もします．そうすると，シーケンサにサブルーチンコールの機能も付けたくなります．通常は数レベルのサブルーチンコールが可能なように設計することが多いようです．

もちろん，ビットスライス型マイクロプロセッサでは，このようなシーケンサもLSI化されて売られていまし

図3



た．

一般のマイクロプログラムでは，今回のような設計とは全然異なり，もっとずっと高速処理に配慮したハードウェアの設計とマイクロプログラムの設計を行います．たとえば，マイクロ命令の実行時間は1クロックですむのに対し，外部の機械命令を読みだすのは数クロック必要なのが普通です．ですから，マイクロプログラムの実行が終わったから次の機械命令を読みだせという信号をマイクロ命令の中に入れておいても，それから機械命令を読みだしていったのではCPUが遊んでしまいます．ですから，メモリアクセスが4クロック必要なら，4クロック後にはマイクロプログラムの実行が終了するから次の機械命令を読み出しはじめろ，という予告信号にしておいて，メモリアクセスと内部の演算が並行して行われるようにするべきです．また，内部でのデータバスが1本だけだと2項演算をするときに2クロック必要だからそれを改良する，ALUが1つだけだとアドレッシング計算と演算を同時に行えないので2つにしておくなど，さまざまな工夫が行われています．

マイクロプログラム方式 その応用例

マイクロプログラムは，普通は特殊なROMに書き込んで(ないしはLSIの内部に作りこんで)使われます．でも，ある種のワークステーションのカスタムCPUのマイクロプログラムは，書き込み可能なメモリにマイクロプログラムが納めてあったりしました．必要に応じて命令体系を変更できるというCPUです．また，昔のミニコンピュータなどでは，マイクロプログラムのROMの一部がオプションで変更できるモデルもたくさんありました．たとえば科学計算用に使う場合はROMを追加することで2進の浮動小数点演算命令が使えるようになったり，事務計算の場合はべつのROMを追加してBCD表示の10進数演算命令が使えるようになったりするものがあります．

ところで，マイクロプログラムでハードウェアをコントロールするのは，なにもCPUに限った話ではありません．たとえばグラフィック描画用のLSI (例：NECのGDC μPD7220) では，直線や円を描画する際に，適切な手順で内部回路を用いて次に描画する点の座標やアドレスを求めなければいけません．このような場合にも，マイクロプログラムの手法が使えます．外部から見れば，いくつかのレジスタに正しくデータを書き込んでやれば，なんらかの期待する動作をするハードウェアに過ぎませんが，内部ではマイクロプログラムが動作している場合というのも，ある程度以上複雑なLSIではよく見られることです．

図4　拡張したマイクロ命令フォーマット

E	A1	A0	RR	RW	Op2	Op1	Op0	W2	W1	RY	JC	ジャンプアドレス（10bit）

JCは条件ジャンプを表すフィールド

表4

E	A1	A0	RR	RW	Op2	Op1	Op0	W2	W1	RY	JC	JA									
0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

表5

E	A1	A0	RR	RW	Op2	Op1	Op0	W2	W1	RY	JC	JA									
0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0

第3回

謎のプロシージャ編

酒井法雄

先月は，構造化制御構文について紹介しました．しかし，これだけではまだまだイカすプログラミングとはいえません．今回はプロシージャを使って，イカすプログラミングに近付きましょう．

謎の円盤UFO

1980年，すでに人類は地球防衛組織SHADOを結成していた．SHADOの本部は，イギリスのとある映画会社の地下深く秘密裏に造られ，沈着冷静なストレイカー最高司令官のもと，日夜謎の円盤UFOに敢然と挑戦していた……．これはイギリスITCの名作，「謎の円盤UFO」のオープニングです．最近，シリーズ半分ほどがLD化されました．この番組が放送されていたころ，毎回欠かさずドキドキしながら見ていた私は，サイフの中身を見てちょっとだけ迷ってからこのLDを買いました．

この作品が製作されたのは，いまから22年前の1970年です．作品中の年代設定は1980年ですから，10年後を想定したSFテレビ映画だったわけです．しかし，1980年からすでに干支もひと回りしてしまいました．当時は心ときめかせて見たものでしたが，いま見ると「ちょっとヘンだな」と思うこともあります．

すでに月面基地があったり，登場人物の服装が妙にSF的だったりします．クルマといえばガルウィングばかりですし，ハデなメカがたくさん出てきます．その割には，コンピュータがやけに古くさかったりします．端末らしい端末もなく，UFOの情報をプリンタで印字したものを持ってきたりします．また，ミサイルの軌道計算にいたっては，無線電話で指示し，パイロットがそれを入力する始末です．

1980年といえば，すでにパーソナルコンピュータがあった時代なのです．コンピュータ文化がこれほど進歩するとは，サンダーバードを作ったITCでも思いも及ばなかったのでしょう．しかし，全般的にはいま見ても心ときめく，いい作品です．

沈着冷静なストレイカー司令官

さて，この作品の主役といえば，やはりストレイカー司令官でしょう．「沈着冷静，ときに非情な男」と作品中では描かれていますが，やはりいま見ると相当強引なオッサンです．しかし，宇宙戦艦ヤマトの古代君ほどバカじゃないので救われています．SHADOの本部には，サンダーバードのように戦闘メカは集中していません．あくまでも司令本部なのです．

UFOの侵入を防ぐのは，まず，月面にあるムーンベースとそこから発進するインターセプターです．コンピュータ衛星SIDとともに，これを第1次防衛網と呼んでいます．ここをUFOが突破してきたときには，第2次防衛網で

あるスカイダイバー艦隊とシャドーモービルが防衛にあたります．

このように，SHADOの組織も手続きをふんだ構造化されたものになっています．この手続きは，構造化制御構文を使って書くことができるでしょう．しかし，これはストレイカー司令官がひとりでできることではありません．司令官は，報告された情報を元に，ムーンベースやスカイダイバー艦隊などに指示を出すだけです．あとは，指示されたSHADOのメンバーが指示にしたがって行動するわけです．ひとりの人間にできることは限られていますから，このほうが能率的なわけです．

この方法を，プログラミングにも応用しようというのが，今回のテーマであるプロシージャです．各部隊の役割を決め，その関係をきちんと構造化した組織にしてしまえば，司令官であるあなたの負担は少なくなります．万が一，UFO，もといバグが侵入しても，問題点を洗いだして改善することもたやすくなるというわけです．

プロシージャとは

このようにプログラミングをラクにしてくれるというプロシージャとは，はたしてどういったものなのでしょうか．ひとことでいうならば，プロシージャとは関数やステートメントのことです．

ステートメントとは，CLSやPRINTなどのようなものです．「PRINT "Hello"」は，「Hello」という文字列を画面に表示する命令です．"Hello"のように，命令の後につけるパラメータを引数(ひきすう)と呼びます．命令をしてやれば，それを実行するだけのものをステートメントと呼びます．

関数は，ASCやCHR$のようなものです．「CHR$ (13)」は，13番目のアスキーコードの文字を返します．戻ってきたものを，戻り値(もどりち)と呼びます．数学でいう関数と同じで，引数を何か入れてやると，なんらかの作業をして，それに対応した戻り値が返ってくるものが，関数です．

もし，PRINTというステートメントがなかったら，ユーザーが最初から作らなくてはいけません．これは，かなり面倒な作業です．そこで，一般的に使われるであろうPRINTやCHR$などのプロシージャは，あらかじめ用意されているわけです．

QuickBASICでは，このようなプロシージャをユーザーが作ることができるのです．もちろん，ステートメントも関数も作ることができます．では，実際にQuickBASICで作れるプロシージャを紹介しましょう．

SUBプロシージャ

ステートメントに相当するのが，SUBプロシージャです．一番簡単なSUBプロシージャは，リスト1のようなものです．

このプロシージャは，「Hello!」という文字列を表示するだけのものです．「SUB Hello」と「END SUB」の間に，やりたい処理を記述すればいいわけです．このように，一般的に「SUB プロシージャ名」の形でプロシージャを定義します．

このHelloプロシージャを使うためには，プロシージャを作ったと宣言しなくてはいけません．ソースの最初の行に，リスト1-1のように記述します．最後の括弧の中が空なのは，引数がないことを示します．

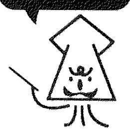
もっとも，QuickBASIC環境版のエディタを使っていると，「SUB Hello」だけ書いてリターンを押せば，自動的に「END SUB」が記述され，そのプロシージャだけが表示されるようになります．[F2]キーでプロシージャを選ぶことができますから，能率的にプログラミングできます．また，ファイルをセーブすれば，自動的に宣言も記述されます．また，セーブしなくてもQuickBASIC環境版では，宣言がなくても実行することができます．

SUBプロシージャHelloを呼び出すには，次のように記述します．ここでは，メインプロシージャ(とくにプロシージャであると宣言しませんが，QuickBASICで最初に実行されるところをメインプロシージャと呼びます)から呼び出します．

```
Hello
```

これだけで，新しく作ったプロシージ

リスト1



```
SUB Hello
    PRINT "Hello!"
END SUB
```

```
DECLARE SUB Hello ()
```

リスト1-1

ワンラインNEWS
日本電池は車に装備されたライター・ソケットを電源に，ノート型パソコンを稼動させるカーアダプタを開発，発売を開始した．主要メーカーのノート型パソコンに対応．

ャを呼び出すことができます.

引数付き SUBプロシージャ

それでは，このプロシージャに引数をつけてみましょう．リスト2を見てください.

ここでは，handleというSTRING型の引数をとることにしました．もし，整数型なら，「SUB Hello (x AS INTEGER)」などのようになります．すでにご存知かもしれませんが，QuickBASICでは「$」がつかなくても宣言してあれば，文字列型の変数として扱うことができます．たとえば，次のufoという変数は文字列型の変数になります.

```
DIM ufo AS STRING
```

プロシージャの引数として定義された変数も，同様に扱われます．したがって，ここでは「handle」という文字列型変数の後ろに「さん」をつけることができるわけです.

このプロシージャを呼び出すには，次のように記述します.

```
Hello "エリス中尉"
```

また，次のように記述することもできます.

```
CALL Hello ("エリス中尉")
```

CALLをつけないときには，括弧もいりません．QuickBASICのステートメントと同じわけです．自分で作ったプロシージャであることを明確にしたいときには，CALLを使うとよいでしょう.

このプロシージャを呼び出すと，つぎのように表示されます.

```
Hello! エリス中尉さん
```

ちなみに，この場合の宣言は，リスト2-1のようになります.

FUNCTION プロシージャ

これで，文字列を引数として，それを編集して画面に出力するSUBプロシージャができました．しかし，編集した文字列は表示するだけでなく，別の用途にも使いたいことがあります．ここでは，編集した文字列を返す関数を作ってみましょう.

さきほどのSUBプロシージャを，リスト3のように書き換えます.

ここで変わっているのは，つぎのような点です.

まず，定義の先頭の「SUB」が，「FUNCTION」になっています．FUNCTIONは，関数であるという意味です．プロシージャ定義の終りも，「END FUNCTION」になっています.

次に，プロシージャ名が「Hello」から「Hello$」になっています．この「$」は，戻り値が文字列型であることを示しています．もし，整数型が戻り値ならば，「%」がつくことになります.

最後に，PRINTしていたところが変わっています．編集した文字列を「Hello$」に代入してあります．「Hello$」はこのプロシージャ名であると同時に，関数の戻り値になります．したがって，プロシージャ名と同じ名前の変数に，戻り値を代入するわけです.

この関数を呼び出すには，次のように記述します.

```
dum$ = Hello$ ("レイク大佐")
```

ここで注意しなくてはいけないのは，FUNCTIONプロシージャは関数であることです．関数は，戻り値をなんらかの形で使わなくてはいけません．したがって，上のように変数に代入したり，別の関数やステートメントの引数にするなどしないと，エラーになってしまいます.

ここでは，文字列型変数dum$に戻り値を代入しています．次のようにすれば，先ほどのSUBプロシージャと同等に編集した文字列を表示することができます.

```
PRINT Hello$ ("レイク大佐")
```

このFUNCTIONプロシージャの宣言は，リスト3-1のようになります.

リスト2

```
SUB Hello (handle AS STRING)
    handle = handle + "さん"
    PRINT "Hello! "; handle
END SUB
```

```
DECLARE SUB Hello (handle AS STRING)
```

リスト2-1

ローカル変数

これで，プロシージャの使い方はだいたいわかりましたね．しかし，実はまだ重要なことが抜けています．先ほどのプログラムをちょっといじって，リスト4のようにしてみます．

なんのことはありません．Hello$プロシージャを呼び出したあとで，handle変数の中身を表示しているだけです．このプログラムを実行すると，次のようになります．

［リスト4の実行結果］

```
Hello! フォスター大佐さん
 0
```

1つめのPRINT文では正常に表示されていますが，2つめのPRINT文では,「0」が表示されています．handleの内容は，Hello$プロシージャの「handle ＝ handle ＋ "さん"」という文で変更されているはずです．なぜ,「フォスター大佐さん」が表示されないのでしょうか?

実は，プロシージャ内の変数は，メインプロシージャやほかのプロシージャの変数とは別扱いになるのです．したがって，Hello$プロシージャでのhandleは文字列型の変数ですが，メインプロシージャではhandleという変数を宣言してないので，単精度の数値型変数になっているわけです．仮に，文字列型の変数にしても，他のプロシージャの同名の変数とは区別されます．このように，プロシージャ内で宣言された変数は，そのプロシージャ内でのみ有効な変数となり，ほかのプロシージャからは参照することができません．このような変数を，ローカル変数と呼びます．

プロシージャとローカル変数をうまく使えば,「あれ? この名前の変数ってどこかで使ってたかな？」などと悩むこともなくなるわけです．

```
FUNCTION Hello$ (handle AS STRING)
    handle = handle + "さん"
    Hello$ = "Hello! " + handle
END FUNCTION
```

```
DECLARE FUNCTION Hello$ (handle AS STRING)
```

リスト3-1

```
DECLARE FUNCTION Hello$ (handle AS STRING)
PRINT Hello$("フォスター大佐")
PRINT handle
END

FUNCTION Hello$ (handle AS STRING)
    handle = handle + "さん"
    Hello$ = "Hello! " + handle
END FUNCTION
```

グローバル変数

しかし，どこのプロシージャからも見ることができ，内容を変更したい変数が欲しいこともあります．このようなときには，グローバル変数を使います．グローバル変数は，メインプロシージャから，次のように宣言します．

```
DIM SHARED sid AS INTEGER
```

ここでは,「sid」という名前の整数型のグローバル変数を宣言しています．DIMのあとの「SHARED」がグローバル変数であるという意味です．これで，すべてのプロシージャからこの変数を参照することができるようになります．ただし，面倒だからといって，何でもグローバル変数にしてはいけません．どこかで変数がかち合って，収拾がつかなくなる恐れがあります．グローバル変数は，必要最小限にしましょう．

また，メインプロシージャで使われているある変数を，特定のプロシージャから見たいということもあります．このようなときには，引数として渡すのが正しいのですが，次のようにサブプロシージャに書く方法もあります．

```
SHARED sky1 AS INTEGER
```

こうすると，そのサブプロシージャからメインプロシージャの変数を参照することができます．

ローカル変数やグローバル変数など，変数が有効である範囲を,「変数の有効範囲」または,「変数のスコープ」

ワンラインNEWS
CSK総合研究所は米タイガーメディア社とマルチメディアソフトの販売，開発で提携．第1弾としてオーサリングツール「CATS MEOW」の国内販売を開始した．

と呼びます．

アドレス渡しと値渡し

少々ややこしくなってきましたが，もうひとつだけお話ししておかなくてはいけないことがあります．リスト5を見てください．

これは，リスト4をちょっと書き換えたものです．メインプロシージャでも文字列型変数handleを宣言し，「ニナ少尉」を代入しています．しかし，先ほどお話ししたように，プロシージャごとに変数はローカルですから，メインプロシージャのhandleとHello$プロシージャのhandleは区別されます．

ところが，このプログラムを実行すると，次のようになります．

[リスト5の実行結果]

```
Hello! ニナ少尉さん
ニナ少尉さん
```

メインプロシージャの2つ目のPRINT文で出力されるのは，その前で「handle = "ニナ少尉"」と代入されているわけですから，「ニナ少尉」のハズです．しかし，「さん」がついています．これは，明かにHello$プロシージャの影響を受けているようです．

実は，プロシージャに渡した変数が渡された先のプロシージャで変更されると，プロシージャから戻ってきてからも影響を受けるのです．メインプロシージャの変数名を別のものにしても，同じ結果になります．

これは，便利なこともありますが，予想しない事態が起きる可能性もあります．仮に，計算をするプロシージャで元の変数の内容が変更されてしまったら，元の値がわからなくなってしまいます．

このようなときには，呼び出し側で次のように記述します．

```
PRINT Hello$ ((handle))
```

引数に2重に括弧がついています．これは，プロシージャに引数のコピーを渡すという意味です．渡したのはコピーですから，プロシージャで内容を変更しても，呼び出し元に戻れば，元の値に影響を与えないわけです．

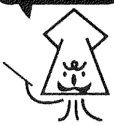
このように，元の引数のコピーを渡す方法を，「値渡し（あたいわたし）」または，「call by value」などと呼びます．一方，引数そのものを渡してしまうことを，「アドレス渡し」「参照渡し」または，「call by reference」などと呼びます．

用途によって使い分けるとよいでしょう．

また，この方法はSUBプロシージャでも同様に使うことができます．

```
DECLARE FUNCTION Hello$ (handle AS STRING)
DIM handle AS STRING

handle = "ニナ少尉"
PRINT Hello$(handle)
PRINT handle
END

FUNCTION Hello$ (handle AS STRING)
    handle = handle + "さん"
    Hello$ = "Hello! " + handle
END FUNCTION
```

リスト5



サンプルプログラム

これで，プロシージャについて一通りの説明をしました．SUBとFUNCTIONの使い分けや，変数のスコープ，引数の渡し方など，今回はなかなか盛りだくさんでした．この文を読んでサンプルプログラムを打ち込むだけでは，完全には理解できないかもしれません．なかなかイメージがわかないこともあるでしょうが，1つ1つのプログラムを作って実験してみると，だんだんわかってくるはずです．

それでは，前回作ったラケットゲームを，プロシージャを使って書きなおしてみましょう．

前回は仕様を決定してから，構造化日本語を元にしてQuickBASICに書きなおしました．構造化制御構文を使うと，手続きを明確に書くことができました．今回は，構造化日本語を，もっと単純にしてみましょう．

図1を見てください．それぞれの構造化日本語の後ろにあるのが，対応するプロシージャです．

リスト6（racket1.bas）がプログラムの全リストです．基本的にプロシージャを使って書きなおしただけですから，前回のプログラムと動作はほとんど同じです．プロシージャを使って

▼リスト6　racket1.bas

```
DECLARE SUB Weight (w AS INTEGER)
DECLARE SUB Display (x AS INTEGER, y AS INTEGER, s AS STRING)
DECLARE SUB DispNumber (x AS INTEGER, y AS INTEGER, s AS STRING, n AS
INTEGER)
DECLARE SUB DispWin (c AS INTEGER)
DECLARE SUB InitScreen (w AS INTEGER, cb AS INTEGER, pb AS INTEGER)
DECLARE SUB WaitCR ()
DECLARE SUB Bells (n AS INTEGER)
DECLARE FUNCTION Game% (dx AS INTEGER, dy AS INTEGER, w AS INTEGER)
DECLARE SUB KeyBufferClear ()
DECLARE SUB DrawItems (x AS INTEGER, y AS INTEGER, px AS INTEGER, cx
AS INTEGER)
DECLARE SUB IsStartOK ()
DECLARE SUB DecBalls (lost AS INTEGER, cb AS INTEGER, pb AS INTEGER)
DECLARE SUB Commands (px AS INTEGER, w AS INTEGER)
DECLARE SUB ComputerMoves (x AS INTEGER, cx AS INTEGER)
DECLARE SUB AdjustY (y AS INTEGER, dy AS INTEGER, sy AS INTEGER)
DECLARE SUB AdjustX (x AS INTEGER, dx AS INTEGER, sx AS INTEGER)
DECLARE SUB IsWall (x AS INTEGER, dx AS INTEGER)
DECLARE SUB Shot (x AS INTEGER, rx AS INTEGER, dy AS INTEGER, sx AS
INTEGER, sy AS INTEGER)

'****************************************
'          Racket Game version 2.0
'       Copyright(c) Norio Sakai 1991
'          All Rights Reserved.
'****************************************

CONST BALL$ = "●"                              'ボール
CONST PAD$ = "========"                        'ラケット
CONST XL = 1                                   'コート左座標
CONST XR = 80                                  'コート右座標
CONST YT = 2                                   'コート上座標
CONST YB = 24                                  'コート下座標
CONST PY = YB                                  'プレイヤーラケットY座標
CONST DF = 500                                 '速度変化単位
CONST XS1 = 10                                 '速度表示X座標
CONST XS2 = 60                                 'ボール数表示X座標
CONST YSP = 25                                 'プレイヤー得点表示Y座標
CONST YSC = 1                                  'コンピュータ得点表示Y座標
CONST CY = 2                                   'コンピュータラケットY座標
CONST MX = 30                                  'メッセージ表示X座標
CONST MY = 12                                  'メッセージ表示Y座標
CONST MESSAGE$ = "Hit [Return] to Start" 'メッセージ

DIM pb AS INTEGER, cb AS INTEGER               'ボール数
DIM w AS INTEGER                               'ウェイト
DIM lost AS INTEGER                            '負けたボールの位置
DIM SHARED pdx AS INTEGER, cdx AS INTEGER'ラケット移動単位
DIM dx AS INTEGER, dy AS INTEGER               'ボール移動単位

'**** 各種変数初期化 ****
pdx = INT(LEN(PAD$) / 2) - 1: cdx = pdx
w = 10000
pb = 5: cb = 5
dx = 1: dy = 1

'**** メインループ ****
DO
    InitScreen w, cb, pb             'スクリーン初期化
    IsStartOK                        'ゲームスタートの確認
    lost = Game%(dx, dy, w)          '1ゲーム
    DecBalls lost, cb, pb            'ボールを減らす
LOOP UNTIL cb = 0 OR pb = 0          'いずれかのボールがなくなるまでループ
DispWin cb
END

'**** 両端の壁当たりボール位置矯正 ****
SUB AdjustX (x AS INTEGER, dx AS INTEGER, sx AS INTEGER)
    x = x + (dx * sx)
    IF x < XL THEN
        x = XL
    ELSEIF XR < x THEN
        x = XR
    END IF
END SUB

'**** 両プレイヤー当たりボール位置矯正 ****
SUB AdjustY (y AS INTEGER, dy AS INTEGER, sy AS INTEGER)
    y = y + (dy * sy)
    IF y < YT THEN
        y = YT - 1
    ELSEIF YB < y THEN
        y = YB + 1
    END IF
END SUB

'**** ボールロストのベル ****
SUB Bells (n AS INTEGER)
    DIM i AS INTEGER

    FOR i = 0 TO n
        BEEP
    NEXT i
END SUB

'**** キー入力処理 ****
SUB Commands (px AS INTEGER, w AS INTEGER)
    SELECT CASE UCASE$(INKEY$)
    CASE "4"                          'ラケット左移動
        Display px, PY, SPACE$(LEN(PAD$))
        px = px - pdx
        IF px < 1 THEN
            px = 1
    END IF
CASE "6"                          'ラケット右移動
   Display px, PY, SPACE$(LEN(PAD$))
    px = px + pdx
        IF 80 - LEN(PAD$) < px THEN
            px = 80 - LEN(PAD$)
        END IF
    CASE "S"                                   '速度を遅くする
        IF w < 32767 - DF THEN
            w = w + DF
            DispNumber XS1 + 6, YSP, "", w
        END IF
    CASE "F"                                   '速度を速くする
        IF 1000 < w THEN
            w = w - DF
            DispNumber XS1 + 6, YSP, "", w
        END IF
    END SELECT
    KeyBufferClear                             'キーバッファのクリア
END SUB

'**** コンピュータラケット移動 ****
SUB ComputerMoves (x AS INTEGER, cx AS INTEGER)
    IF x < cx THEN                             'ボールが左にあるとき
        Display cx, CY, SPACE$(LEN(PAD$))
        cx = cx - cdx
        IF cx < 1 THEN
            cx = 1
        END IF
    ELSEIF cx + LEN(PAD$) < x THEN   'ボールが右にあるとき
        Display cx, CY, SPACE$(LEN(PAD$))
        cx = cx + cdx
        IF 80 - LEN(PAD$) < cx THEN
            cx = 80 - LEN(PAD$)
        END IF
    END IF
END SUB

'**** ボールの数を減らす ****
SUB DecBalls (lost AS INTEGER, cb AS INTEGER, pb AS INTEGER)
    IF lost < YT THEN
        cb = cb - 1
    ELSE
        pb = pb - 1
    END IF
    Bells 10
END SUB

'**** 汎用文字列表示 ****
SUB Display (x AS INTEGER, y AS INTEGER, s AS STRING)
    LOCATE y, x
    PRINT s;
END SUB

'**** 汎用数値表示 ****
SUB DispNumber (x AS INTEGER, y AS INTEGER, s AS STRING, n AS INTEGER)
    LOCATE y, x
    PRINT s;
    PRINT USING "#######"; n;
END SUB

'**** 勝ち判定 ****
SUB DispWin (c AS INTEGER)
    IF c = 0 THEN
        Display MX, MY, "You Win!"
    ELSE
        Display MX, MY, "Computer Win!"
    END IF
END SUB

'**** ボールとラケットの描画 ****
SUB DrawItems (x AS INTEGER, y AS INTEGER, px AS INTEGER, cx AS
INTEGER)
    Display x, y, BALL$                        'ボール描画
    Display px, PY, PAD$                       'プレイヤーラケット描画
    Display cx, CY, PAD$                       'コンピュータラケット描画
END SUB

'**** 1ゲームループ ****
FUNCTION Game% (dx AS INTEGER, dy AS INTEGER, w AS INTEGER)
    DIM px AS INTEGER, cx AS INTEGER           'ラケットX座標
    DIM sx AS INTEGER, sy AS INTEGER           'ボール移動係数
    DIM x AS INTEGER, y AS INTEGER             'ボール座標

    px = 40 - (LEN(PAD$)): cx = px
    sx = 1: sy = 1
    x = 40: y = 7                              'ボール初期位置

    DO UNTIL y < YT OR YB < y                  'ボールがはみ出たら次の面
        Weight w                               'ウェイト
        Display x, y, "  "                     'ボールの消去
        IF (y <= YT AND cx <= x AND x < cx + LEN(PAD$)) THEN
            Shot x, cx, dy, sx, sy             'コンピュータのショット
        ELSEIF (y >= YB AND px <= x AND x < px + LEN(PAD$)) THEN
            Shot x, px, dy, sx, sy             'プレイヤーのショット
        END IF
        IsWall x, dx                           '壁の当たり判定
        AdjustX x, dx, sx                      'X方向補正
        AdjustY y, dy, sy                      'Y方向補正
        ComputerMoves x, cx                    'コンピュータのラケット移動
        Commands px, w                         'プレイヤーコマンド処理
        DrawItems x, y, px, cx                 'ラケットとボールの描画
    LOOP
    Game = y
END FUNCTION

'**** 面初期化 ****
SUB InitScreen (w AS INTEGER, cb AS INTEGER, pb AS INTEGER)
    CLS
    DispNumber XS1, YSP, "Speed ", w
    DispNumber XS2, YSC, "Balls :", cb
    DispNumber XS2, YSP, "Balls :", pb
```


ワンラインNEWS
アドバンテストは，社内報の印刷をパソコン通信で印刷会社に発注するオンラインシステムを構築した．社内報の作成が効率的になり，通信費も節約できるのが特徴．

```
END SUB

'**** ゲームスタートの確認 ****
SUB IsStartOK
    Display MX, MY, MESSAGE$
    WaitCR
    Display MX, MY, SPACE$(LEN(MESSAGE$))
END SUB

'**** 両端の壁当たり判定 ****
SUB IsWall (x AS INTEGER, dx AS INTEGER)
    IF x <= XL OR XR <= x THEN
        dx = -dx
    END IF
END SUB

'**** キーバッファ消去 ****
SUB KeyBufferClear
    DO UNTIL INKEY$ = ""
    LOOP
END SUB

'**** ショット ****
SUB Shot (x AS INTEGER, rx AS INTEGER, dy AS INTEGER, sx AS INTEGER,
sy AS INTEGER)
    BEEP
    dy = -dy
    IF x = rx + INT(LEN(PAD$) / 2) THEN                    'スーパーショット
        sx = 2: sy = 4
    ELSEIF x = rx OR x = rx + LEN(PAD$) - 1 THEN           'ヘンテコショット
        sx = 2: sy = 1
    ELSE                                                   'ノーマルショット
        sx = 1: sy = 1
    END IF
END SUB

'**** リターンキー待ち ****
SUB WaitCR
    DO UNTIL INKEY$ = CHR$(13)
    LOOP
END SUB

'**** ウェイト ****
SUB Weight (w AS INTEGER)
    DIM i AS INTEGER

    FOR i = 0 TO w
    NEXT i
END SUB
```

書いたプログラムでは，前回よりもわかりやすくなっているだけでなく，能率的になっています．

たとえば，任意の位置への文字列の表示をプロシージャにしたので，あちこちに「LOCATE」と「PRINT」を書く必要がなくなっています．また，Shotプロシージャは，引数を変えて2回呼び出しています．このルーチンは，コンピュータとプレイヤーのショットの判定です．前のプログラムでは2回書いていたわけですが，プロシージャにして同じルーチンを使うようにしたので，もしショットの判定を変えたいときにも1か所の変更で済ませることができます．

また，グローバル変数は，本来は定数と同じ扱いにしたいもののみにしていることにも注目してください．本当にあちこちのルーチンで必要になるのは，定数です．必要ない限り，グローバル変数は使わないようにしましょう．

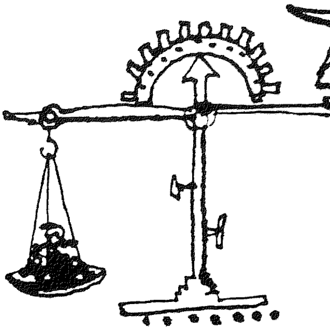
ほかにも，いろいろと注目してほしいところがあります．前回のリストと見比べて，どのようにすればわかりやすいプログラムが作れるかを，じっくりと考えてください．

次回は，変数や定数の命名法など，プログラムをわかりやすくする方法について述べます．それでは，また来月お会いしましょう．

▼図1

```
プロシージャの宣言
定数の初期化
変数の宣言
変数の初期化
DO
    画面の初期化……InitScreen
    スタートOKか?……IsStartOK
        リターン待ち……WaitCR
    1ゲーム……Game%
        DO UNTILボール位置がラケットの後ろになったら
            ウエイト処理……Weight
            打つ……Shot
            横の壁に当ったか……IsWall
            X方向矯正……AdjustX
            Y方向矯正……AdjustY
            コンピュータラケットを動かす……ComputerMoves
            コマンド処理……Commands
                キーバッファ消去……KeyBufferClear
            ボールとラケットの描画……DrawItems
        LOOP
    負けた方のボールを減らす……DecBalls
        ベルを鳴らす……Bells
LOOP UNTILいずれかのボールがなくなる
勝者を表示する……DispWin
END


汎用ルーチン
    文字列表示……Display
    数値編集表示……DispNumber
```

Illustration : Masami Nozue



アルゴリズム篇 第3回

サーチ・その1

実践C塾

三代目塾頭
(株)ジャステックラボ
加野島英渡／山本博昭

前回まで，文字列のソート(並べ替え)のアルゴリズムについて考察してきたが，今回からソートと並んでもっとも使用頻度の高いアルゴリズムである，検索について考えてみたい．今回取り上げるのは，「番人」を用いる方法と「学習機能」を利用する方法である．

先月までのあらすじ

データ処理におけるさまざまなアルゴリズムを紹介しながら，実際のプログラムに応用していくということでスタートしたこの連載，はじめの2回でアルゴリズム界の代表選手であるソートを考察してきた．バブルソートからクイックソートへのアルゴリズム変更による劇的な効果は記憶に新しいことと思うが，かようにアルゴリズムはプログラムにおけるデータ処理の効率に大きな影響を及ぼすものなのである．

そして，このアルゴリズムというものは「いかなる状況においても絶対」というものではなく，処理の対象となるデータの量・構造などによってその状況に適したものは変わってくる．さまざまな状況下において，必要な処理の特性を把握し，適切なアルゴリズムを選択することができるようになるためには，いろいろなアルゴリズムとその基本的な考え方を理解しておくことが必要になる．

今月の重点項目

さて前回までは，単語リストが完成したあとでの表示部分の処理（単語をコード順にソートして表示する）を見てきたわけだが，今回からはこのテーマツールをまた別の観点から見なおしてみることにする．今度は表示の前段階である単語リストを作る過程での処理に注目することにしてみよう．

テーマツールにおいて，ファイルから単語を切り出してリストを作る処理は，以下の2つに大別される．

①その単語が初出の単語かどうか調べる［リストの検索］
②初出であれば新しいリストを作り，既出であればカウンタを更新する．［リストの更新］

①の処理は，いわゆる「検索」処理である．検索はソートと並んでもっとも使用頻度の高いアルゴリズムであろう．エディタ・ワープロをお使いの読者なら，文字列の検索は日頃からお馴染みであろうし，よく見かける「where」というツール（目的のファイルがディスク上のどこのディレクトリにあるかを表示する）も検索の一例である．ほかにも夕日を探して海岸を走るという森田検索などというのを考えついたが，これは職場などでは用いないほうが賢明であろう……．

テーマツールでの検索は，「単語リストの中から目的の文字列がすでに登録されているかどうか調べる」処理をするものである．連続した領域に格納されている長文の中から，ある単語を検索するのと異なり，単語リストというインデックスをもったデータ構造の中を検索していく処理であるから，効率をあげるためのアルゴリズムも変わってこよう．今回から，この検索アルゴリズムを中心に，いかにしたらより効率よく単語リストが検索・更新できるかというところにテーマツールの強化ポイントをおくことにしよう．

検索について

これから検索のためのアルゴリズムをいくつかあげていくが，最初はテーマツールのデータ構造をそのまま使った場合の検索方法を考察していくことにする．

あるデータ群の中から，目的のものを探すことを検索と呼ぶ．探すとはいっても，「対象となるものと一致するものを探す」，「対象となるものと一致しないものを探す」，「この範囲にあてはまるものを探す」，「この範囲から外れているものを探す」といったように目的は種々ある．

また，データ群の構造によっても検索アルゴリズムは変わってくる．ある構造（ソート済みであるなど）をもったデータ構造ならば，もっと効率のよい別の検索アルゴリズムを使用することも可能だが，現状でのテーマツールの単語リストのようなランダムなデータに対しては，先頭から順に1つ1つ検索していくということになる．順次検索していくので，この手法はシリアルサーチと呼ばれる．

シリアルサーチのさまざまな手法

【正直に検索する方法】

テーマツールの単語リストは，ファイルをリードしていって，単語を出現順に順次リスト化していくので，そのデータ並びはランダムになっている．このため，現在のプログラムでは，あ

る単語が初出のものかどうかをチェックするために，単語リストの先頭から正直に文字列比較していく方法をとっている．図1がそのプログラミングである．

このプログラムを抽象化して書くと図2のようになる．

運よく最初のほうで一致する文字列が見つかればいいが，そうでない場合は大半の時間を左側のほうのループを回るという処理に費やしてしまうことになる．このループが終了する条件は

- **一致する文字列が見つかった**
- **ポインタがデータ列の末尾になった**

という2つであり，繰り返しのたびに毎回この2つの条件のいずれかが真になっていないかをチェックすることになる．

【「番人」を置く方法】

上記の処理は，開始する際のちょっとした工夫でもっと簡単なプログラム構造に改造することが可能である．比較しようとする文字列をストッパーとして単語リストの末尾に書き込んでおいてから比較を開始するのである．こうしておけば，最悪でも単語リスト末尾にいま書き込んでおいた文字列があるので，「絶対に見つかる」ということになる．この保証があれば，リストが末尾になったどうかのチェックをループのたびにする必要はなくなる．

このストッパーのことを「番人」と呼んでいる．「番人」を使った検索のアルゴリズムをさきほどと同じように書くと図3のようになる．

ループの出口が1つになったことがわかる．ポインタが末尾をさしているかどうかはループを抜けたあとで1度チェックするだけになる．チェックの結果，ポインタが末尾をさしているという場合は，処理開始時に入れておいた「番人」が働いたということなので，データ群の中に目的のものが見つからなかったことを意味している．この手法による変更は，全体としては小さなものではあるが，ループの脱出条件を

図1　全部正直に検索するロジック

```
  : static     BOOL     wexist(char *sptr)
比較する単語のポインタを受け取り，発見できたかどうかの真偽を返す．

  : struct     _wlist     *wlp;
いまどこを探しているかのワーク用ポインタ

  :      for(wlp = wlist ; wlp < wlptr ; wlp++)
  :      {
単語リストの先頭から最終位置まで探す

  :           if(strcmp(wlp->wbody , sptr) == 0)
比較する文字列と現在のポインタ先の文字列との比較を行う

  :           {
  :                wlp->wcount++;
  :                return    BTRUE;
  :           }
もし一致したら，カウンタを＋1して真で終了

  :      }
  :      return    BFALSE;
最後まで一致するものがなかったら偽で終了
```

図2

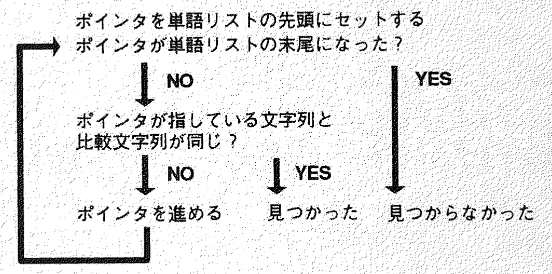
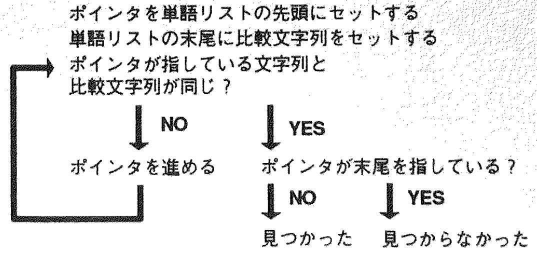


図3



ワンラインNEWS
日電アネルバはシリコンウェハー表面に金属皮膜をを形成するスパッタリング装置の精度を大幅に高める技術を開発．これにより256MビットDRAMの量産も容易に．

シンプルにするということによって繰り返しのたびに行うチェックが減るため，意外に大きな効果を得ることができる場合がある．

【学習機能を付加する方法】

いままでの手法で，データ群の先頭から後方に向かって検索を行うと，同じものを何回探しても，必要な処理時間はつねに一定ということになる．検索対象のキーデータの出現頻度にある程度のかたよりがあるような処理を何度も行う場合には，これを改良して，データ群の中から目的のものが見つかったときに，そのキーデータをデータ群の先頭のほうに配置しなおすという処理を付け加えることがある．こうすることによって，同じキーを次回に検索するときにはより早く目的のデータを発見できるようになる．

このことですぐに連想されるのは，日本語入力の「辞書学習機能」であろう．もっとも，最近の日本語入力ソフトがもっている辞書の学習機能はこのような単純なものではなく，複雑なアルゴリズムによってその優先順位を更新しているのだと思われる（このアルゴリズムが日本語入力の「賢い」，「○○だ」という評価に密接に結び付いているような気がする）．

それでは，検索の際のデータの学習効果をいくつか考えてみよう．いずれも基本的な考え方は，いま見つかったデータをデータ群の前のほうに配置しなおすということである．

①見つかったデータと先頭のデータを交換する．

この手法は，図4のように同じデータを連続して検索する場合には大きな効果がある．また，先頭のデータと発見したデータを入れ替えるだけなので，必要なロジックはごくシンプルなものである．ただし1度でも別のデータを検索した場合には，それまで先頭にあったデータは最悪ならデータ列の最後方になってしまうので，同じデータを何度も検索する保証がない限り効果がないことになる．

②見つかったデータを先頭のデータとし，以降のデータを1つずつずらしていく．

発見されたデータが次回の優先順位1位となる点では，さきほどの手法と同じであり，1度出現したデータに対しては，すぐに大きな効果が得られるうえ，次に別のデータを検索してもそのデータの順位は1つ下がるだけである．

結果的に出現頻度の高いデータが前のほうに固まってくるので，‘D’→‘C’→‘D’→‘C’などという検索をしても効果が持続する．ただ，処理のたびにデータの移動が伴うので，インデックスやポインタリストをもったデータ構造を対象にしなければ繁雑ではある．

③見つかったデータとその1つ前のデータを交換する．

図4

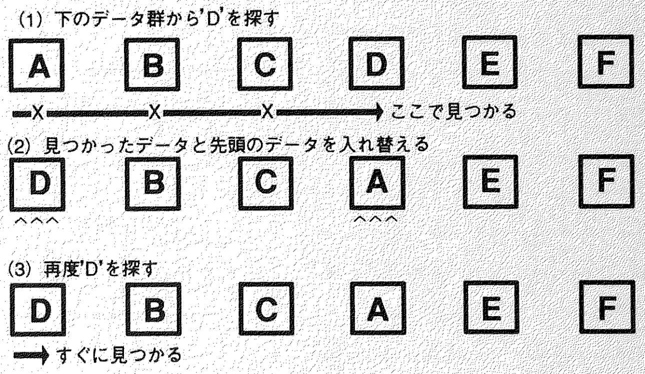
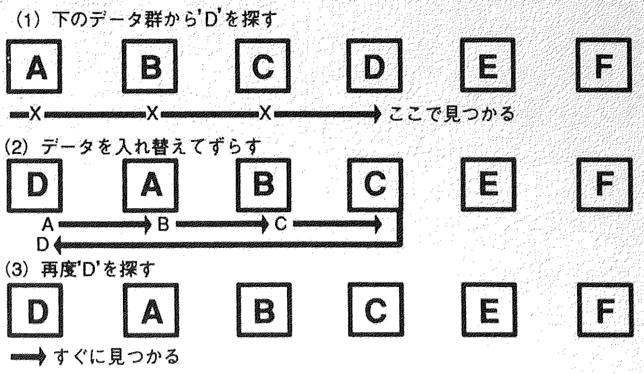


図5



データが発見されるたびに少しずつ優先順位があがってくるので，すぐに大きな効果は得られないが，出現する頻度の高いデータを検索するほど効率があがってくる．処理自体は，②の手法と違って，直前のデータと交換するだけなので，①と同じように簡単である．

④見つかったデータを自分より頻度が高いものが出てくるまで上げる

③の手法を少し改造したものである．発見したデータを直前のデータと入れ替えるという基本的な発想は同じだが，それまでの各データの出現頻度によって入れ替えが発生するかどうかを決定する．データを発見すると，無条件に入れ替えを行わずに，直前のデータと出現カウントを比較する．もしいま発見したデータのほうが出現カウントが大きくなっていればデータを入れ替える．さらにもう1つ前のデータとカウントを比較していき，そのデータより出現カウントの大きなものが出てくるか，データ列の先頭に達するまで交換を繰り返す．この手法では③の以下のような欠点が補強される．

- たまたま出てきた出現頻度の低いデータによって出現頻度の高いデータが後ろに追いやられる可能性がある．
- どのデータも同じ条件で前に移動していくので，出現頻度の高いデータが前に行くのに時間がかかる場合がある．

ただし，④の改造によって以下のような弱点も発生する．

- カウントを比較しながらデータを次々と交換していくので処理は多少複雑になる．

ところでいま，この手法が③の欠点を補強すると書いたが，実はどんな状況においてもこの手法のほうが優れているわけではない．論より証拠，簡単なツールを作ってこの2つの検索手法を検証してみることにする．

【2つの学習機能をツールを作って検証する】

検索の結果をデータ群の順位に反映させるアルゴリズムの効率を簡単なツールを作って検証してみることにする．作成したツールの仕様は，「指定されたディレクトリから下のすべてのディレクトリを検索して，特定の拡張子がいくつあるかカウントする．」というものである．検索の対象となる拡張子は「.TXT」，「.DOC」，「.BAT」，「.COM」,「.C」,「.H」,「.OBJ」,「.EXE」の8つとした．各ディレクトリでそれぞれの拡張子をもつファイル名をカウントしていき，最後に結果を表示する．同時に検索に要したループ回数とデータの優先順位を変えるために発生した

図6

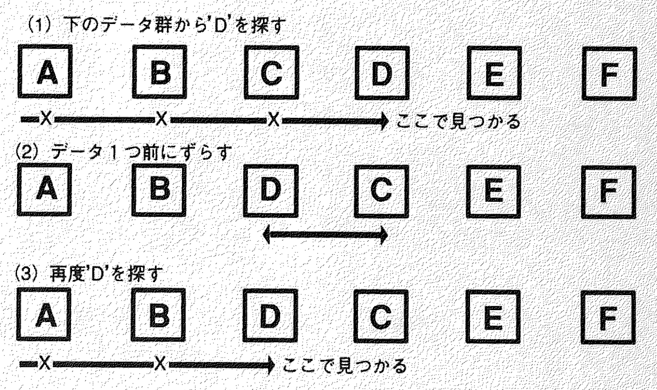
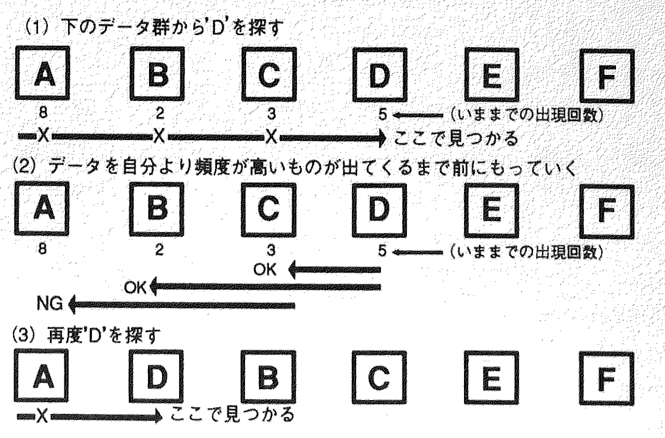


図7



ワンラインNEWS
ライフボートは，MS-DOSをFA向けに改変できるソフト「RTx86」と「RT386」の販売を開始した．これによりリアルタイム・マルチタスクシステムが構築できる．

入れ替えの回数も表示する．

ツールの起動方法は

A> SRCHEXT 〈検索開始ディレクトリ名〉

とする．〈検索開始ディレクトリ名〉を省略した場合はカレントディレクトリからの検索となる．3種類の検索方法を用意したので，実行する場合はリスト2に示したメインのソースの指定箇所にリスト3～5のうちの該当ファイルをはめ込んでコンパイルしていただきたい．コンパイルするためのMAKE用ファイルはリスト1にある．

例によってBorland C++で作成したが，たいていのCコンパイラでコンパイルできるはずである．

以降に出てくる図は，それぞれのアルゴリズムをはめ込んだプログラムを筆者のパソコンで実行した画面であ

リスト1 SRCHEXT.MAK

```
.AUTODEPEND

#        *Translator Definitions*
CC = bcc +SRCHEXT.CFG
TASM = TASM
TLINK = tlink

#        *Implicit Rules*
.c.obj:
  $(CC) -c {$< }

.cpp.obj:
  $(CC) -c {$< }

#        *List Macros*

EXE_dependencies =  ¥
  SRCHEXT.obj

#        *Explicit Rules*
SRCHEXT.exe: SRCHEXT.cfg $(EXE_dependencies)
  $(TLINK) /x/n/c/P-/LA:¥BC¥LIB @&&|
c0s.obj+
SRCHEXT.obj
SRCHEXT
        # no map file
emu.lib+
maths.lib+
cs.lib
|

#        *Individual File Dependencies*
SRCHEXT.obj: SRCHEXT.c

#        *Compiler Configuration File*
SRCHEXT.cfg: SRCHEXT.mak
  copy &&|
-ff-
-vi
-IA:¥BC¥INCLUDE
-LA:¥BC¥LIB
-P-.C
| SRCHEXT.cfg
```

リスト2 SRCHEXT.C

```
/* ================================================================

    検索アルゴリズムチェック用サンプル "Srchext.c"
                  Copyright 1991 by JUSTECH-LABO. (H.kanoshima)

================================================================ */

#include    <stdio.h>
#include    <string.h>
#include    <dos.h>
#include    <dir.h>

#define     PATHNAMELEN     80

struct  _ExtTable
{
    char    *ExtName;
    int     HitCount;
}ExtTable[] =
    {"TXT", 0,
     "DOC", 0,
     "BAT", 0,
     "COM", 0,
     "C"  , 0,
     "H"  , 0,
     "OBJ", 0,
     "EXE", 0,
     NULL, 0};

long    SearchLoopCount;
int     SwapCount;


/* ↓↓↓この部分に SRCH1 SRCH2 SRCH3 のいずれかをはめ込んでください ↓↓↓ */

/* ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑ */


void    DisplayExtAndCount(void)
{
int     i;

    for(i = 0; ExtTable[i].ExtName != NULL; i++)
        printf("%3s = %4d¥n", ExtTable[i].ExtName, ExtTable[i].HitCount);

    printf("¥nCount = %ld¥n", SearchLoopCount);
    printf("¥nSwapCount      = %d¥n", SwapCount);

}

void    SearchExt(void)
{
char    NewDta[256];
char    far *SaveDta;
struct  ffblk   FileInformation;
char    StartPath[PATHNAMELEN];

    printf("%s¥n", getcwd(StartPath, PATHNAMELEN));

    SaveDta = getdta();
    setdta(NewDta);

    if(findfirst("*.*", &FileInformation, FA_DIREC) != 0)   return;

    do
    {
        if(*FileInformation.ff_name == '.')     continue;

        if(FileInformation.ff_attrib == FA_DIREC)
        {
            chdir(FileInformation.ff_name);
            SearchExt();
            chdir("..");
        }
        else
        {
            CountExt(FileInformation.ff_name);
        }
    }while(findnext(&FileInformation) == 0);

    setdta(SaveDta);
}

void    main(int argc, char **argv)
{
char    StartPath[PATHNAMELEN];

    SearchLoopCount = 0;

    getcwd(StartPath, PATHNAMELEN);

    if(argc == 2)
        chdir(argv[1]);

    SearchExt();

    DisplayExtAndCount();

    chdir(StartPath);
}
```

る．筆者の使用環境は20MBのハードディスクで，サブディレクトリも13個と比較的小さなシステムである(「翔泳社」というディレクトリがあるが，この記事のためにわざわざ作成したのではなく，毎号の原稿を書くために元々使っていたことをご理解願いたい)．「普通，ハードディスクって180MBくらいあるよね」と豪語する読者なら，もっと異なる傾向が出てくることと思う．

図8が，SRCH1を用いた普通の検索手法での実行画面である．結果は，検索終了時のデータ列の並びをそのまま表示しているので，データの入れ替

リスト3　SRCH1

```
/* ************************************************************ */
/*  SRCH1                                                       */
/*     データ構造を変化させない検索手法                         */
/* ************************************************************ */

void    CountExt(char *FileName)
{
char    *Extp;
int     i;

    Extp = strchr(FileName, '.');
    if(Extp == NULL)    Extp = "";
    else                Extp++;

    for(i = 0; ExtTable[i].ExtName != NULL; i++)
    {
        if(strcmp(ExtTable[i].ExtName, Extp) == 0)
        {
            ExtTable[i].HitCount++;
            break;
        }
        else
        {
            SearchLoopCount++;
        }
    }
}
```

図8　SRCH1によるSRCHEXT.EXE実行

```
B>SRCHEXT ¥
B:¥
B:¥翔泳社
B:¥翔泳社¥サンプル
B:¥翔泳社¥原稿
B:¥文書
B:¥SOURCE
B:¥SOURCE¥ASM
B:¥SOURCE¥C
B:¥STOCK
B:¥STOCK¥BAKLOG
B:¥TOOL
B:¥TOOL¥BIN
B:¥TOOL¥DOC
B:¥TRASH
TXT =    77
DOC =    18
BAT =    13
COM =    40
  C =    39
  H =     9
OBJ =    32
EXE =   127

Count = 7102

SwapCount        = 0
```

リスト4　SRCH2

```
/* ************************************************************ */
/*  SRCH2                                                       */
/*     出現したデータの優先順位をひとつずつ上げていく検索手法   */
/* ************************************************************ */

void    SwapData(void *x , void *y , int width)
{
char    *cptrx = x;
char    *cptry = y;

    while(width--)
    {
    char    tmp;

        tmp = *cptrx;
        *cptrx++ = *cptry;
        *cptry++ = tmp;
    }
}

void    CountExt(char *FileName)
{
char    *Extp;
int     i;

    Extp = strchr(FileName, '.');
    if(Extp == NULL)    Extp = "";
    else                Extp++;

    for(i = 0; ExtTable[i].ExtName != NULL; i++)
    {
        if(strcmp(ExtTable[i].ExtName, Extp) == 0)
        {
            ExtTable[i].HitCount++;
            if(i != 0)
            {
                SwapData(&ExtTable[i], &ExtTable[i - 1],
                          sizeof(struct _ExtTable));
                SwapCount++;
            }
            break;
        }
        else
        {
            SearchLoopCount++;
        }
    }
}
```

リスト5　SRCH3

```
/* ************************************************************ */
/*  SRCH3                                                       */
/*     出現したデータの優先順位を自分より頻度の高いものが出て   */
/*     くるまで上げていく検索手法                               */
/* ************************************************************ */

void    SwapData(void *x , void *y , int width)
{
char    *cptrx = x;
char    *cptry = y;

    while(width--)
    {
    char    tmp;

        tmp = *cptrx;
        *cptrx++ = *cptry;
        *cptry++ = tmp;
    }
}

void    CountExt(char *FileName)
{
char    *Extp;
int     i;

    Extp = strchr(FileName, '.');
    if(Extp == NULL)    Extp = "";
    else                Extp++;

    for(i = 0; ExtTable[i].ExtName != NULL; i++)
    {
        if(strcmp(ExtTable[i].ExtName, Extp) == 0)
        {
        int     Revi;

            ExtTable[i].HitCount++;
            Revi = i;
            while(Revi)
            {
                if(ExtTable[Revi].HitCount > ExtTable[Revi - 1].HitCount)
                {
                    SwapData(&ExtTable[Revi], &ExtTable[Revi - 1],
                              sizeof(struct _ExtTable));
                    SwapCount++;
                }
                else    break;
                Revi--;
            }
            break;
        }
        else
        {
            SearchLoopCount++;
        }
    }
}
```


ワンラインNEWS
ロジテックは米アップル社のMacintoshシリーズに接続できる縦型HDD「LHD-M100S」を開発，関東電子が販売する．

えのないこの処理ではソースリストにある配列のままであることを確認していただきたい．「Count = 7102」が示すように350個ほどのファイルの拡張子を分類するのに7000回以上のループが必要になっている．データ列の変更は行わないので，当然「SwapCount = 0」である．

図9は，発見するたびにそのデータを1つ前に進めるSRCH2の場合である．「Count = 6091」となり，何と1000回以上も参照回数が減っている．この減少に貢献しているのが「SwapCount = 178」というデータの入れ替えである．結果を見ると「.EXE」と「.TXT」が群を抜いて多い(これはあくまでも筆者の使用している環境での話である)．処理の経過を見ることはできないが，検索途中のデータ配列ではこの2つの拡張子データが出現のたびに前のほうに移動していき，配列の先頭のほうに位置しているはずである．

また，表示の順番が検索終了時の状態を示しているが，必ずしも出現数の多い拡張子が配列の先頭にあるわけではないのは，検索の最後のほうに出てきた拡張子が前に来ているためである．たとえば，「DOC = 18」がデータ配列の先頭に来ているのは，「B:¥TOOL¥DOC」というディレクトリを最後のほうで検索したためであると思われる．

図10が，SRCH3を用いたプログラムの実行画面である．この手法では出現頻度をチェックしながらデータを交換していくので，終了時のデータ配列はカウントの大きい順にきれいに並んでいることがわかる．また，1度出現頻度の高いデータが上のほうにくると，入れ替えが発生しなくなってくるので「SwapCount = 52」と，SRCH2の単純に1つ前と入れ替えるアルゴリズムより交換の回数がたいへん少なくなっている．ところが，ループの回数は「Count = 6341」とSRCH2と比べて逆に増えている．交換も少なくなっているのだから，検索効率はいちばんいいという予想だったはずだが，事実は小説よりも奇なり……である．そこで，この現象を解析して検索の効率について考えてみることにしよう．

【学習機能の実際の効果について】

ここでもう一度，SRCH2とSRCH3の手法の違いを確認してみよう．SRCH2はデータが発見されるたびに確実にそのデータがデータ配列の中で1つ前のほうに配置される．これに対して，SRCH3では発見したデータの前方にどんな出現頻度をもったデータがあるかによって，配置のされ方が変化してくる．例をあげてみると

例1：すべてのデータが同じ出現カウント状態である検索開始当初に，結果的に出現頻度が低かったデータを多く検索した場合は，SRCH3ではそのデータが一気にデータ列の先頭に配置され，後半に出現した頻度の高いデータは先に先頭となったデータの出現数を超えるまで前に配置することができない．

SRCH2の手法では，データは出現数の分しか前に進まないので，どういう順番でデータが到着しても頻度の高いデータはそれだけ前進して効率がよくなる．

例2：逆の場合はSRCH3が圧倒的に効率がよくなるだろう．つまり，最終的に出現頻度の高かったデータが検索当初からコンスタントに出現すると，常にデータ列の先頭に位置するように

図9　SRCH2によるSRCHEXT.EXE実行

```
B>SRCHEXT ¥
B:¥
B:¥翔泳社
B:¥翔泳社¥サンプル
B:¥翔泳社¥原稿
B:¥文書
B:¥SOURCE
B:¥SOURCE¥ASM
B:¥SOURCE¥C
B:¥STOCK
B:¥STOCK¥BAKLOG
B:¥TOOL
B:¥TOOL¥BIN
B:¥TOOL¥DOC
B:¥TRASH
DOC =    18
TXT =    77
BAT =    13
COM =    40
EXE =   127
  C =    39
OBJ =    32
  H =     9

Count = 6091

SwapCount        = 178
```

図10　SRCH3によるSRCHEXT.EXE実行

```
B>SRCHEXT ¥
B:¥
B:¥翔泳社
B:¥翔泳社¥サンプル
B:¥翔泳社¥原稿
B:¥文書
B:¥SOURCE
B:¥SOURCE¥ASM
B:¥SOURCE¥C
B:¥STOCK
B:¥STOCK¥BAKLOG
B:¥TOOL
B:¥TOOL¥BIN
B:¥TOOL¥DOC
B:¥TRASH
EXE =   127
TXT =    77
COM =    40
  C =    39
OBJ =    32
DOC =    18
BAT =    13
  H =     9

Count = 6341

SwapCount        = 52
```

なり，SRCH2に比べて処理ループもデータ交換も少ない検索になる．

筆者の場合の実行画面を見て想像がつく読者も多いと思うが，ディレクトリ検索の順番の関係で最初に処理される「翔泳社」というディレクトリには原稿などのテキストファイル（世間の大半の通例に従って筆者も.TXTという拡張子をつけている）が多く，最終的に参照数の多かった.EXEファイルは，後半で検索される「TOOL」というディレクトリにその大半が格納されている．つまり，先ほどの例1のデータ出現パターンに近いということになる．そしてこのパターンは，前述したようにSRCH3の手法が苦手とするものであった．このように，検索するデータの出現パターンによっても適するアルゴリズムは変化してくる．

もう1つ考慮に入れなければならないのは，検索ループとデータの交換に要する処理量の問題である．せっかく処理ループの回数を減らしても，そのためのデータ交換に大きな手間がかかっていれば（長い文字列の実体を交換するというような），何にもならない．逆に，検索に非常に手間がかかるようなデータ構造であれば，データ交換に多少手間がかかっても，検索回数を減らすことによって大きな効果を得る場合もある．

今月は検索の手法をいろいろと考察してきた．ここではいつものように，それらのうちのいくつかを実際に使ってテーマツールを改造することにする．

まず改造に先立って，その効果を検証するためのロジックを先月号までのプログラムに加えておく．検証は「検索に要した処理ループの総回数」によって行い，検索のアルゴリズムを変更することによって，処理ループの回数がどのように変化するかを実際の数値で確かめてみることにする．

先月号のソース「WCSUB.C」の中の関数「wexist」と「dump_word」をリスト6，7のものと入れ替えてコンパイルし直していただきたい．変更点は，単語検索のロジックにカウンタを入れて1回のループのたびに増加させていき，結果を表示するときに，その時点でのカウンタ値を表示するというものである．

この改造版wcount.exeを使って，先月と同じように「WCOUNT.C」をカウントさせると，最終行の表示は

```
loop count total=2267
```

となる．単語リストの検索のために関数wexistのfor文で2267回のループを行ったという意味である．

では，いよいよアルゴリズムの変更に移ることにしよう．先ほど考察した2つの手法「出現するたびにリストの順位を1つずつ上げていく」，「番人を置く」というアルゴリズムをテーマツールに取り入れてみる．

出現するたびにリストの順位を1つずつ上げていく

前述したように，データ群の優先順位を発見のたびに1つずつ上げていくというアルゴリズムのコーディング自体は簡単なものである．

テーマツールの場合だと，関数「wexist」の中で，既出の単語が発見されるたびにその単語のリストを直前のものと交換すればよい．もちろん，その単語がリストの先頭であれば交換は行わない．図11のリスト中にある「=====」で囲まれた部分がその追加部分である．

リスト6　先月号のwcsub.cのチェック用ロジック追加1

```
long loop_total = 0;

static  BOOL    wexist(char *sptr)
{
struct  _wlist  *wlp;

    for(wlp = wlist ; wlp < wlptr ; wlp++, loop_total++)
    {
        if(strcmp(wlp->wbody , sptr) == 0)
        {
            wlp->wcount++;
            return  BTRUE;
        }
    }
    return  BFALSE;
}
```

リスト7　先月号のwcsub.cのチェック用ロジック追加2

```
void    dump_word(void)
{
struct  _wlist  *wlp;
    printf("¥n**** ソート中 ****¥n");
    sort_data();
    for(wlp = wlist ; wlp < wlptr ; wlp++)
        printf(" (%04d) - '%s'¥n" , wlp->wcount , wlp->wbody);
    printf("loop count total=%lu¥n", loop_total);
}
```


ワンラインNEWS
日本電気と日立製作所は音声データを統合したLAN（=IVD LAN）の普及で協力する．93年頃から標準規格に基づく製品を商品化していく．

ここまでの改造で先ほどと同じ様に実行した結果は

```
loop count total=2254
```

となる．改造前が2267回ということで，wcount.cのような非常に小さなソースでも効果が認められる．このソースの場合，単語数が59個でそのうち2回以上出現する単語が11個，単語の出現回数は最大でも4回ということで効果もあまり大きくないが，出現回数の多い単語を含むような大きなデータだと効果も大きくなる．たとえば，今月のwcsub.cで試してみると37284回：36439回ということで，効果が大きくなっていることがわかる．

図11　wexist関数に「出現するたびにリストの順位を1つずつ上げていく」改造

```
: static    BOOL    wexist(char *sptr)
: {
: struct    _wlist    *wlp;
: char    *cptr;
:
:     for(wlp = wlist ; wlp < wlptr ; wlp++,loop_total++)
:     {
:         if(strcmp(wlp->wbody , sptr) == 0)
:         {
:             wlp->wcount++;
================================================================
:             if(wlp != wlist)
:                 swap_data(wlp,wlp-1,sizeof(struct _wlist));
================================================================
まず，現在のポインタ(wlp)が単語リストの先頭(wlist)であるかどうかチェックする．
先頭でなければ，現在のポインタの1つ前との入れ替えをswap_dataを使って実行する．

:             return    BTRUE;
:         }
:     }
:     return    BFALSE;
: }
```

番人を置く

次に「番人を置く」というロジックを追加してみる．前述したように，このロジックは，これから検索する対象となるキーをあらかじめデータ群の中に仕込んでおいてから検索を開始する手法である．こうしておけば，絶対にキーが発見されるので，データ群が終わりになったかどうかをチェックすることなしに検索ループの処理を継続することが可能になる．

さらにキーがデータ群の中に含まれていなかった場合，番人として仕込んだデータがそのままデータ群の最新のものとして次回から利用できる．このことから，テーマツールへの改造は検索関数「wexist」および単語リストへの単語登録関数「wset」の2つに及ぶことになる．図12，13のリストがそれ

COLUMN　再帰的にディレクトリ検索を行う関数

今回のテーマである「検索」とは少々性質は異なるが，実験プログラムのために作成したディレクトリ検索関数を紹介したい．

この関数は，カレントディレクトリから下の全ディレクトリに対して再帰的に移動していって必要な処理を行うものである．呼び出す前にルートディレクトリに移動しておけば，そのディスク内のすべてのディレクトリ中のファイルに対する処理を行うことも可能になる．ディレクトリ移動処理は関数内で閉じているので，汎用性もあると思う．ぜひとも自作のツールに応用していただきたい．リストaにこの関数の動作を解説する．

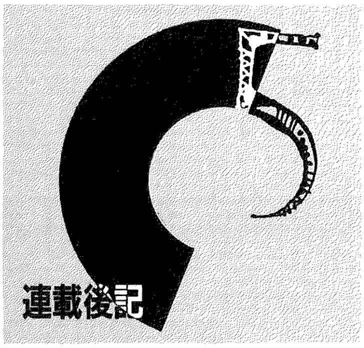
リストa

```
: void    SearchExt(void)
: {
: char    NewDta[256];
: char    far *SaveDta;
: struct    ffblk    FileInformation;
: char    StartPath[PATHNAMELEN];
:
:     printf("%s¥n", getcwd(StartPath, PATHNAMELEN));
処理中のサブディレクトリ名を表示する。

:     SaveDta = getdta();
別のディレクトリでfindfirstを実行すると現在のＤＴＡが更新されるので保存しておく

:     setdta(NewDta);
ローカルエリアに新しいＤＴＡを設定する。

:     if(findfirst("*.*", &FileInformation, FA_DIREC) != 0)    return;
「最初のファイル検索」でファイルがなければ、何もしないで終了

:     do
:     {
:         if(*FileInformation.ff_name == '.')        continue;
ファイル検索では「.」と「..」も発見されるので、この2つは処理をスキップしなければならない。

:         if(FileInformation.ff_attrib == FA_DIREC)
見つかったのがディレクトリ名だったら・・・
:         {
:             chdir(FileInformation.ff_name);
そのディレクトリ名へ移動する。

:             SearchExt();
再帰的に自分を呼び出して処理を行なって

:             chdir("..");
元の位置に戻る

:         }
:         else
見つかったのがファイルだった場合

:         {
:             CountExt(FileInformation.ff_name);
※ここに各ディレクトリ内のファイルに行ないたい処理を書く

:         }
:     }while(findnext(&FileInformation) == 0);
「次のファイル検索」でファイルが見つからなくなるまで繰り返す

:     setdta(SaveDta);
ＤＴＡを処理開始時の場所に戻す
: }
```

それの関数の変更概要である．

「番人」を仕込むことによって，「wexist」のループ処理がfor文から単純な無限ループになった．ループ1回ごとにwlp < wlptrを行っていたのを，キー文字列が見つかった時点で1度だけ行うようになったためである．

今月も，関数の内容を入れ替えるだけの改造なので，実行ファイルの作成や起動方法は第1回目のものとまったく同じである．リスト8にあるサブソース「WCSUB.C」のみ入れ替えて，メインソースの「WCOUNT.C」およびMAKEファイルも第1回目のものをそのままお使い願いたい．



今回は「検索」をテーマに順次検索のアルゴリズムを考察した．現時点でのテーマツールのデータ構造では，単語リストがランダムに組成されるため，順次検索をせざるをえないが，データ構造そのものを見なおすことによって別のアルゴリズムを導入することも可能である．

次回はデータ構造の見なおしまで踏み込んで別の検索手法を紹介することにしたい．

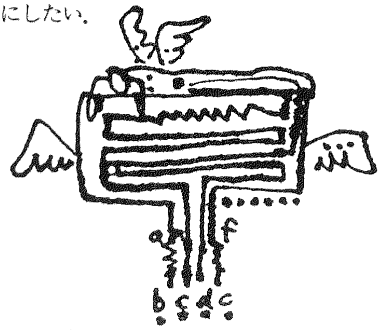


図12　wexist関数に「番人」を置く改造

```
: static     BOOL     wexist(char *sptr)
: {
: struct     _wlist     *wlp;
: char     *cptr;
:
:      cptr = (char *)malloc(strlen(sptr) + 1);
:      if(cptr == NULL)
:      {
「番人」を格納するためのメモリをアロケートする．

:           printf("作業用メモリが足りません¥n");
:           exit(1);
メモリが確保できない場合はプログラムを終了する．

:      }
:      wlptr->wbody = strcpy(cptr , sptr);
単語リストの末尾に「番人」を仕込む

:
:      wlp = wlist ;
検索は単語リストの先頭から始める

:      while(1) {
必ず一致するものがあるので，単純な無限ループとしてよい．

:           if(strcmp(wlp->wbody , sptr) == 0) {
一致する文字列が見つかった場合

:                if(wlp < wlptr ) {
ポインタがリスト末尾に達していなければ，既出の単語である．

:                     wlp->wcount++;
単語のカウンタを更新する．

:                     if(wlp != wlist)
発見した箇所が単語リストの先頭でなければ...

:                          swap_data(wlp,wlp-1,sizeof(struct _wlist));
単語の優先順位をひとつ上げる．

:                     free(cptr);
既出の場合は「番人」を廃棄する．

:                     return     BTRUE;
見つかったので真を返す

:                }
:                else {
「番人」のところで見つかった場合は初出の単語である．

:                     wlptr->wcount = 1;
:                     wlptr++;
「番人」を新しい単語リストに正式に加える処理

:                     return     BFALSE;
初出なので偽を返す

:                }
:           }
:      wlp++ ;
ポインタの更新

:      loop_total++ ;
ループカウンタの更新

:      }
: }
```

図13　「番人」のロジック追加に伴うwset関数の改造

```
: void     wset(char *sptr)
: {
:      if(wexist(sptr) == BTRUE)
:      {
:           putchar('!');
真が返ってきた場合は，他に何も処理がない

:      }
:      else {
:           if(wlptr >= &wlist[WORDMAX+1]) {
:                printf("単語数が %d 個を超えました．¥n",WORDMAX);
:                exit(1);
偽が返ってきた場合は，「wexist」内で新しい単語が登録されている．
前回まではここで単語の登録を行なっていたが，仕込んだ「番人」がそのまま
新しいリストになるのでここでの処理はしなくてもよいことになる．ただし，
単語リストがいっぱいになったかどうかの処理はここで行なうことにする．

:           }
:           putchar('.');
:      }
: }
```


ワンラインNEWS
富士ソフトウエアはパソコンに接続した日本語OCRシステム向けのパッケージソフトを開発．近く販売を開始する．業種，帳票の書式の種類にかかわらず対応できる．

リスト8 WCSUB.C

```c
/* =================================================================

    WSUB.C          Copyright 1991 by JUSTECH-LABO. (H.kanoshima)

================================================================== */
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>

#define     WORDMAX     1000

typedef     enum    {BFALSE,BTRUE}  BOOL;

struct  _wlist{
    int     wcount;
    char    *wbody;
}wlist[WORDMAX+1];

struct  _wlist  *wlptr = wlist;

/* ===============================================================

============================================================== */
void    swap_data(void *x , void *y , int dwidth)
{
char    *cptrx = x;
char    *cptry = y;

    while(dwidth--)
    {
    char    tmp;

        tmp = *cptrx;
        *cptrx++ = *cptry;
        *cptry++ = tmp;
    }
}
/* ===============================================================



============================================================== */
/* 検索ループの総回数をカウントするワーク */
long    loop_total=0;

static  BOOL    wexist(char *sptr)
{
struct  _wlist  *wlp;
char    *cptr;

    /* 「番人」のための領域を確保する */
    cptr = (char *)malloc(strlen(sptr) + 1);
    if(cptr == NULL)
    {
        printf("作業用メモリが足りません¥n");
        exit(1);
    }
    /* 「番人」を組み込む */
    wlptr->wbody = strcpy(cptr , sptr);

    wlp = wlist ;
    while(1) {
        if(strcmp(wlp->wbody , sptr) == 0) {
            if(wlp < wlptr ) {
                wlp->wcount++;
                /* リスト先頭でなければ順位を一つあげる */
                if(wlp != wlist)
                    swap_data(wlp,wlp-1,sizeof(struct _wlist));
                /* 「番人」を廃棄する */
                free(cptr);
                return  BTRUE;
            }
            else {
                /* 「番人」をデータとして使う */
                wlptr->wcount = 1;
                wlptr++;
                return  BFALSE;
            }
        }
    wlp++ ;
    loop_total++ ;
    }
}

/* ===============================================================


============================================================== */
void    wset(char *sptr)
{
    if(wexist(sptr) == BTRUE)
    {
        putchar('!');
    }
    else {
        if(wlptr >= &wlist[WORDMAX+1]) {
            printf("単語数が %d 個を超えました。¥n",WORDMAX);
            exit(1);
        }
        putchar('.');
    }
}
/* ===============================================================


============================================================== */
int     compar(const void *x , const void *y)
{
const struct _wlist *wptrx = x;
const struct _wlist *wptry = y;

    return  strcmp(wptrx->wbody , wptry->wbody);
}
/* ===============================================================


============================================================== */
int     (*q_compar)(const void * , const void *);
int     q_dwidth;

void    _qusort(char *initial_leftp , char *initial_rightp)
{
char    *centerp , *leftp , *rightp;
int     cmp;

    if(initial_rightp <= initial_leftp + q_dwidth)
    {
        if(initial_rightp == initial_leftp) return;

        if((*q_compar)(initial_leftp  , initial_rightp) > 0)
            swap_data(initial_leftp , initial_rightp , q_dwidth);
        return;
    }

    leftp  = initial_leftp;
    rightp = initial_rightp;
    centerp = leftp;

    do{
        while((*q_compar)(leftp  , centerp) < 0)    leftp  += q_dwidth;
        while((*q_compar)(rightp , centerp) > 0)    rightp -= q_dwidth;

        cmp = leftp - rightp;
        if(cmp <= 0)
        {
            if(cmp != 0)
            {
                swap_data(leftp , rightp , q_dwidth);
            }
            leftp  += q_dwidth;
            rightp -= q_dwidth;
        }
    }while(leftp <= rightp);

    if(initial_leftp < rightp)
        _qusort(initial_leftp , rightp);
    if(leftp < initial_rightp)
        _qusort(leftp , initial_rightp);
}

/* ===============================================================


============================================================== */
void    sort_data(void)
{
char    *i_left = (char *)wlist;

    q_compar = compar;
    q_dwidth = sizeof(struct _wlist);

    _qusort(i_left , i_left + ((wlptr - wlist - 1)  * q_dwidth));
}
/* ===============================================================


============================================================== */
void    dump_word(void)
{
struct  _wlist  *wlp;
    printf("¥n**** ソート中 ****¥n");
    sort_data();
    for(wlp = wlist ; wlp < wlptr ; wlp++)
        printf("(%04d) - '%s'¥n" , wlp->wcount , wlp->wbody);
    printf("loop count total=%u¥n",loop_total);
}
/* ===============================================================


============================================================== */
char    *wget(char **sptrptr)
{
static  char    getw[2048];
char    *sptrs , *sptrd;
int     cplen;

    sptrs = *sptrptr;

    sptrs += strspn(sptrs , " ,¥t¥n");
    if(*sptrs == '¥0')  return  NULL;

    sptrd = strpbrk(sptrs , " ,¥t¥n");
    if(sptrd == NULL)   cplen = strlen(sptrs);
    else                cplen = sptrd - sptrs;
    sptrd = getw;
    while(cplen--)
        *sptrd++ = *sptrs++;
    *sptrd = '¥0';
    *sptrptr = sptrs;

    return  getw;
}
```

98活用アセンブラ技法

SCSIドライバ編 第3回

ディスクイメージダンププログラムの作成

星野 操

前回までの2回にわたって，デバイスドライバのスケルトンとこのスケルトンを利用したPC-9801の55ボード用SCSI-BIOSをダミーデバイスドライバとして実現した．さらにこのダミーデバイスドライバを利用したSCSIテープドライブ用ドライバも作成した．
今回は，やはりこのダミーデバイスドライバを利用したディスクイメージダンププログラムを作成する．

前回までのプログラム

前回までに，デバイスドライバを作成するためのスケルトン，またそれを利用した98の55ボード用のSCSI-BIOSをデバイスドライバの形でインプリメントしました．これは基本的には，SCSIコントロールルーチンだけをもつ「ダミーの」デバイスドライバで，そのエントリベクタのアドレスを取得するためにIOCTLを利用していました．同様なことは常駐プログラムでも実現できますが，このIOCTLが利用できるということはデバイスドライバを使った常駐ルーチンにおいての利点の1つです．

さらに，それを利用して動作するSCSIタイプのテープドライブのドライバを作成しました．これはキャラクタデバイスとして動作し，テープからの読み込み・テープへの書き込みを通常のDOSコールで実現するものでした．キャラクタデバイスですから基本的には1文字単位でのアクセスを可能としなければいけませんが，実際にはテープは512バイトなどのブロック単位で読み書きされます．ですから，このブロッキング・デブロッキングについてはデバイスドライバの中でやってやらないといけません．

もっとも，実際には1文字単位で読み書きを実行するとオーバーヘッドが大きすぎ，テープの転送速度が低下するのであまり現実的ではありません．ですから，実用的に使用するにはDOSのファンクション3FH/40Hなどの連続した読み書きを，512バイトの整合性をとった上で行う必要があります．

このようにキャラクタデバイスとしてテープデバイスをインストールできれば，Tarなどを利用することによってファイルのバックアップを実行することができます．しかし，Tarはディスクをファイルイメージとして取り扱うため，どうしてもスピードが低下してしまいます．

98のSCSIインターフェース

これはとくに98の遅いSCSIインターフェースの影響が出ているといえます．テープドライブの記録スピードは，前回記したように最高で約6MB/minです．これは決して遅いスピードではありませんが，100KB/secですからさほど速いスピードでもありません．SCSIバスは非同期で1.5MB/secはいきますから，まったく問題にならない程度のスピードです．

ちなみに，普通のSCSI(2ではない)でも同期転送をすれば最高4MB/secの転送速度を出すことができます．非同期と同期の違いは，データの先送りを許可するか否かの違いで，たとえば通信プロトコルでパケットの先送りを許可するかどうかというのと似ているといえるでしょう．小容量(100MB程度)で安いドライブだと非同期転送しかサポートしていない場合がありますが，最近の200MB程度以上のドライブだと，たぶん同期転送をサポートしていることでしょう．もっとも，本体のインターフェースが同期をサポートしていない場合には初期のネゴシエーションで非同期転送になるようになっています．

ハードディスク自体の記録速度は，10～20MB/minですから1～2.5MB/sec程度です．ですから，基本的にはSCSIバスの転送速度が上がっても，さほどトータルの転送時間が短くなるわけではありません．ディスクの記録速度のほうがボトルネックになる可能性があるのです．

ただし，ディスクドライブがある程度の大きさの先読みバッファを備えており，かつマルチタスクシステムなどで複数のSCSIデバイスを同時に使用できるときには大きな意味があります．つまり，あるサイズのデータの読み書きが完了するまでの時間は変わりませんが，バスの転送速度が速くなると，データの転送に必要な時間が短くなるので，転送完了するまでの間に別のデバイスに転送データを送ることができるのです．たとえば，テープからディスクにリストアを実行している場合に，ディスクアクセスの空き時間にテープへのデータ転送を行う，ということが可能になります．

これに対して，DOSのようなシングルタスクシステムだとこのようにはいきません．SCSIバスは空いていても，転送が完了するまで次のデバイスアクセスは許容されませんから，複数のデバイスのコンカレントな動作は不可能となります．つまり，ディスクの転送が完全に終了してから，テープのアクセスを開始する必要が出てくるのです．もっともそのために，テープデバイスドライバなどがかなり簡単に書ける利点はあるのですが．これがマルチタスクシステムだと非常に厄介になります．

98の内部バス転送速度の問題

さてDOSなどのシングルタスクOSの場合は，マルチタスクの場合に比べてさほどSCSIバス自体の転送速度が効いてきません．ところが，98の場合にはそれ以前に内部バスの転送速度の問題が出てきてしまうのです．98は

ワンラインNEWS
米インテルは80386の価格を35%引き下げた．値下げ幅は製品によって異なるが，20MHzの386SXは82.5ドルから59ドルに．同社は世代交替による値下げと説明

SCSIバスの読み書きに，DMAを利用しています．周知のように，98のDMAは非常に遅いのです．1つはDMAチップ自体が低速であるという問題があります．もともと8080のファミリーとして作られたチップが設計の元となっているわけですから，現在の32bit-CPUに比べると遅くてしようがありません．また拡張スロットのI/Oをアクセスする必要があることから，その分のウェートを見込まないといけません．

加えてもっとも悪いのは，98のDMAがシングルトランスファーモードというモードを利用している点です．DMAチップ8237には，転送に3つのモードが存在します．すなわち，規定数のDMA転送が終了するまでバスを専有するモード，要求が連続してある場合にのみバスを専有し続けるモード，そして1バイト転送するごとにバスを解放するモードです．この中でもっとも効率的なのは，要求がある場合にバスを専有し続けるデマンドモードなのです．とくに外部転送速度がDMA速度より遅く，かつ適切なバッファリングがペリフェラルの内部で行われる場合には最高です．

ただし，この場合にはDMA要求の「先読み」が必要になるので，場合によってはDMAオーバーランがおきる可能性があります．ですからペリフェラル側で要求を出すタイミングを少し考えないといけません．さらに以前の98ではDMAを利用してメモリのリフレッシュを行っていたので，その間隔を保証するためにも毎回バスを開放するシングルトランスファーモードが採用されたのです．

毎回バスを開放するということは，毎回バスを獲得しなければいけないということであり，このオーバーヘッドがDMAチップの遅さに加わって，98のSCSIはたいへんに遅いものとなってしまっています．

テープにバックアップをする場合には，データをディスクから読み込み，これをテープに書き出すわけですから，SCSIバスをデータが往復することになります．ですから，転送速度が倍問題になってくるわけです．結果として，98では6MB/secのテープを駆動するのもぎりぎりの速度しか出ないということになってしまいます．

ディスクイメージでのバックアップ

もともとぎりぎりなところに，ファイルイメージでのアクセスをすると，FATの読み込みやその他のDOSフォーマットの処理に時間がかかってしまい，テープにまにあわないことになります．実際，前記のようにTarを使うとよくても実効の転送速度が半分になってしまいます．

これを避けるためには，ディスクイメージをそのままダンプするのがいちばん簡単です．ただ単に連続してディスク内容を読んでいけばよいのですから，アクセスも高速になります．フロッピーベースでのバックアップツールでも，ファイルイメージでバックアップするよりはディスクイメージでのバックアップのほうが高速だったように思います．

ディスクイメージでのバックアップの問題は，次のようになるでしょう．

①つねにディスクの全イメージをアクセスすることになるので，使用量の少ないドライブでは効率が悪く時間がかかる．
②システムなど別のソースから復元可能な部分までもバックアップするので効率が悪い．
③変更されたファイルだけをバックアップするインクリメンタルバックアップが不可能である．
④ファイル単位での復元が不可能である．
⑤不良セクタがあった場合に，処理が困難になる．
⑥同一の諸元のドライブにしか復元することができない．

上記の内，①～③については全体をバックアップする際の効率の問題です．①についてはしかたのない部分もありますが，これらはディスクのパーティションを適切に切ることでかなり回避できます．たとえば，②はシステムのドライブとワークのドライブを分離することで対応できます．③についても適切な間隔でドライブ全体のバックアップをとり，それからのインクリメンタルなバックアップをテープに行う，ということで対処できるように思われます．ただし，インクリメンタルバックアップはTarではできませんから（findfなどと組み合わせる必要がある），なんらかのツールを用意することが必要になります．

なお，システムの部分のバックアップというのは嫌われますが，実際にfatalなトラブルでディスクイメージを復元しようという場合には，時間がないことも多いものです．このようなときにいちいちコンパイラなどのディスクを抜き差しして，再インストールをするのは非常にめんどうです．インストールしたとしても，その後のカスタマイズはまた別に行わないといけません．たとえオリジナルフロッピーがあったとしても，それから環境を整えるための労力は半端ではありませんか

ワンラインNEWS
ソニーは米アップル・コンピュータからパソコンプロトコルのライセンスを受ける．これによりソニーの「NEWS」とアップルの「Macintosh」の接続が可能になる．

ら,やはりシステムも(できれば一括して)バックアップしておくにこしたことはありません.

④については，基本的にどうしようもありません．必要ならディスク全体をリストアしてから，目的のファイルだけを別に抜き出し，移すことになります．このために，実際のワークドライブのほかにリストアができるだけのドライブスペースが欲しくなります．たとえば，同一容量のドライブBとCがあった場合に，まずCをディスクイメージでバックアップし，そこにBのバックアップをリストアしてから目的のファイルを取り出し，最後にCをリストアする，というようなことが可能となります．

ハードディスクのセクタ不良

⑤は大きな問題です．実際，FDベースのバックアップツールは不良セクタがある場合には対応できないものがありました．たとえこれらがマーキングできたとしても，復元するドライブがバックアップしたドライブとつねに同一とは限りませんから，別なところにエラーが生じてしまうかもしれません．

ただ現在のまともなSCSIハードディスクは，みなエラーフリーになっています．これは別に製造技術が上がったということではなく，代替セクタを利用できるようになっているからです．SASIのドライブでも，コントローラに代替トラックという機能はありました．つまりエラーのあるトラックを予備トラックに移し替え，この代替処理をコントローラ側で自動的に行うことにより，見かけ上エラーフリーにしようというものです．ただし，代替トラックは通常トラックの内周部分とられるので，この領域をアクセスするとシーク作業が必要になり速度が低下するという問題があります．加えてコントローラはエラー情報を記憶するための不揮発なメモリをもっていませんから，エラー情報はパワーオンするたびに設定し直してやらないといけません．

このため98では代替トラック処理は使用せずに，FORMATコマンドのサーティファイでエラーが発見された場合には，FATに不良セクタの印を付けることで対処していたのです．このためHDの出はじめのころは不良セクタの多少が問題にされたりもしたものです．

現在のSCSIドライブの大部分は，コントローラ内部でエラー情報を管理します．SCSIドライブは物理的諸元に縛られなくなったことから，その一部をこれらエラー管理情報に割り当てることで，不揮発な管理テーブルを確保することができるようになったのです．また代替トラック以前に代替セクタという方法をとっています．各トラックあるいは各シリンダ単位であらかじめ代替セクタを予約しておくのです．不良が出たらこれにふり替えるわけです．容量的には不利な方法ですが，シーク動作が必要ではないため高速なふり替え処理ができるようになります．万一代替セクタがたりなくなった場合には，別に内周に予約してあるセクタを利用するようになっているものも多いようです．結果として，見かけ上SCSIドライブはまったくエラーフリーになっています．

UNIXなどのシステムは，ドライブは基本的にエラーフリーである必要があります．DOSのようにエラーのある部分を別に予約する，というようなことはほとんど行いません．すべては周辺のほうで管理される必要があるのです．このような要求から，SCSIコントローラの仕様が決まってきたような面も感じられます．

結論として，SCSIドライブではエラーフリーであることが基本であり，エラーがあるようなドライブは「不良」ですからこれについては考慮する必要はあまりないといえるでしょう．ですからこの前提に基づけば⑤はあまり問題にならなくなります．

BPB情報の書き換え

⑥は，同一のドライブに復元する場合はなんら問題にはなりませんが，ドライブ自体が故障してしまった場合などには困ります．同一のドライブが手にはいるかどうかわからないからです．1つ救いであるのは，98のディスク管理の場合は「より大きなパーティションに内容を復元することは可能である」ということです．ハードディスクは，その先頭にパーティションテーブルをもっています．これで1つのドライブを複数のパーティションに分割しているわけですが，それぞれのパーティションはどのOSで使用するかということは確定していません．ですからOSは，パーティションテーブルの情報を直接には使用しません．

たとえばDOSですと，各パーティションの最初のセクタに論理ドライブの管理情報が書かれています．DOSではBPBという形式でドライブの諸元を把握しますが，これそのものが記録されているのです．スタートアップのときには，これら論理ドライブの最初のセクタを読み込み，そこからBPBをドライバのワークエリアにコピーしてきてシステムに渡すのです．

ですから，このBPB情報も含めて全

体を書き換えてしまうと，それがパーティション情報と異なっていてもDOSは正常に論理ドライブを認識します．もちろん，あとから書かれた情報がパーティションの割り当てより大きいとよその部分に上書きをしてしまうため他の論理ドライブを破壊してしまいます．このようなことをしてはいけませんが，小さいドライブの情報を書き込む分にはとくに問題はありません．ただし，正常に認識させるためには1度リセットして再びシステムを立ち上げる必要があります．そのままだと前の管理情報が残ってしまっているからです．

このようなやり方は緊急避難的なので，適当な内容を取り出したあとは元の状態に戻してやらないといけません．このためには，先頭セクタを含めたオリジナルなディスクの内容をオーバーライトしてやるか，あるいはFORMATコマンドで1度領域解放をしてから再び領域確保すればよいことになります．

いずれにしろ，このようなことができることを考えると，確保する領域は少しでも余裕をとって，たとえば128MBぎりぎりでなく127MBぐらいにしておくほうが便利だということになります．

今月のプログラム

ディスクイメージダンププログラムddump

それでは，ディスクのイメージをそのままダンプするプログラムを示すことにしましょう．これは元々はCで書かれていたもので，ライブラリを使用するとかなり短いプログラムになっていました．しかし，この連載でCのプログラムだけを掲載するわけにもいかないですから，全部アセンブラに書き直してみました．もっとも，かなりの部分がCのライブラリに相当する記述になってしまっています．

プログラムddumpは，ディスクイメージをテープデバイスにダンプするプログラムです．オプションはr／f／mの3つがあり，ハイフンの後に記述します．rは復元を意味するもので，これを指定するとテープからリストアを行います．これを指定しないデフォルトでは，バックアップ動作を行います．

fは対象となるデバイス名を指定するもので，デフォルトはstapeです．別の名前のデバイスを使用する場合には，-fNAMEという形で指定します．場合によっては，ディスクのダンプイメージをファイルにすることも可能です．mオプションは，バックアップのときに簡単なコメントを指定できるようにするものです．やはり-mMESSAGEの形で記述しますが，@setupマクロの制限によりメッセージ本体にはホワイトスペースを含ませることはできません．これは""で括った場合には1つの引数と解釈できるように書き直したほうがよいでしょう．バックアップ時に記録されたコメントは，リストアするときに出力されます．

プログラムの解説

最初に，連載のいちばん最初のころに掲載された3つのユーティリティマクロをインクルードします．今回はファイルアクセスも行うので，dosfile.incもインクルードしているわけです．

【ヘッダ構造】

次に，構造体の宣言があります．まずあるのは，記録されたイメージのヘッダ構造です．ディスクのイメージダンプに先立って，バックアップを実行した日時とセクタ数，それからmオプションで指示されたコメントを記録しておきます．これをリストア時に出力させることで，復元しようとしているイメージが求めるものであるかどうかを判断します．時間は，Cのtime関数で得られるものと同様，1970／1／1からの経過時間を秒で示したものを使用します．longサイズの領域が必要です．最初Cで記述されたことからこのような仕様になっていますが，本来は年・月・日　時・分・秒で分割して記述したほうが，プログラムの構造はずっとシンプルになります．今回のプログラムのかなりの部分が，このUNIX形式の時間とのコンバートに使用されています．

【ブロック数】

次のcountは，イメージのテープのブロック数，つまり512バイト単位のブロック数を示します．IBMシステムではブロックデバイスのセクタサイズは，基本的に512バイトに統一されているので，この論理セクタ数と物理セクタ数は同一ですが，98では1KBあるいは2KB／セクタですから，この論理セクタ数は物理セクタ数の2／4倍になります．物理セクタ数は制限により64K個以下ですが，論理セクタ数は16ビットでは表現しきれないことがあります．よってdwordのサイズを確保しています．

【コメント領域】

その次のhcommentはコメントを記録する領域です．これはあらかじめ0でクリアしておきます．

ワンラインNEWS
日本アイ・ビー・エムは三菱重工業神戸造船所に橋梁建設シミュレーション用に「POWERステーション」を納品．建設現場でEWSが使われることはめずらしい．

リスト1 ddump.asm

```
        dosseg
        .model  small,pascal

        include condx.inc
        include utils.inc
        include dosfile.inc

        .stack  1000h

;---------------------------
; structure definition
;

_header struc
        htime   dd      ?
        count   dd      ?
        hcomment        db      128 dup (0)
_header ends

_dpb    struc
        drive_number    db      ?
        logical_number  db      ?
        byte_per_sector dw      ?
        sector_per_cluster      db      ?
        shift_value     db      ?
        reserved_sector dw      ?
        fat_number      db      ?
        directory_number        dw      ?
        top_sector      dw      ?
        total_cluster   dw      ?
        sector_per_fat  db      ?
        director_top_sector     dw      ?
_dpb    ends

;---------------------------
; macro definitions
;

@putc   macro   arg
        dos     <2,,,arg>
        endm

@crlf   macro
        @putc   0dh
        @putc   0ah
        endm

@print  macro   arg
        @string <arg>
        @puts   dx
        endm

@exit   macro   code
        mov     al,code
        dos     <4ch>
        endm

;---------------------------
; work area
;
        .data
drive   dw      ?
fflag   dw      ?
farg    dw      ?
mflag   dw      ?
marg    dw      ?
rflag   dw      ?

header  _header <>

        .data?
buffer  db      8000h dup (?)

;---------------------------
; utility functions
;

        .data
day_of_year     dw      -1,0,31,59,90,120,151,181,212,243,273,304,336,365
day_of_year2    dw      -1,0,31,60,91,121,152,182,213,244,274,305,337,366

        .code
func    time    <time_var>
        ; get time
        dos     <2ch>

        ; ch:hour, cl:minute, dh:second
        mov     al,ch
        mov     ch,dh
        mov     ah,0
        mov     bx,60
        mul     bx
        add     al,cl
        adc     ah,0
        mul     bx
        add     al,ch
        adc     ah,0
        adc     dx,0

        ; save it
        mov     bx,time_var
        mov     [bx],ax
        mov     [bx+2],dx

        ; get date
        dos     <2ah>

        ; cx:year, dh:month, dl:date
        ; calc day of year
        mov     bl,dh
        mov     bh,0
        add     bx,bx
        test    cx,3
        @if     z
                mov     ax,day_of_year2[bx]
        @else
                mov     ax,day_of_year[bx]
        @endif
        add     al,dl
        adc     ah,0

        ; day from 1970 1/1
        xchg    ax,bx
        sub     cx,1970
        mov     ax,365
        mul     cx
        inc     cx
        shr     cx,1
        shr     cx,1
        add     ax,cx

        ; total days
        add     ax,bx
        dec     ax

        ; day->second
        mov     dx,3600*12
        mul     dx
        add     ax,ax
        adc     dx,dx

        ; calc second
        mov     bx,time_var
        add     [bx],ax
        adc     [bx+2],dx

        ; and return
        mov     ax,time_var
        ret
endf

        .data
ctime_buffer    db      40 dup (?)
month_name      db      'Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec '
day_name        db      'Sun Mon Tue Wed Thr Fri Sat '


        .code
func    set_value       <value,separator>
        mov     ax,value
        mov     bl,10
        div     bl
        add     al,'0'
        stosb
        xchg    ah,al
        add     al,'0'
        stosb
        mov     ax,separator
        cmp     ax,-1
        @if     nz
                stosb
        @endif
        ret
endf

func    ctime   <time_var>,<si di>
        local   year,month,date,hour,minute,second
        local   tsecond_of_day,day_offset

        ; load time value
        mov     bx,time_var
        mov     ax,[bx]
        mov     dx,[bx+2]

        ; split value
        mov     second,ax
        and     second,1
        shr     dx,1
        rcr     ax,1
        mov     bx,3600*12
        div     bx
        mov     tsecond_of_day,dx
        mov     day_offset,ax

        ; get year
        mov     bx,365*4+1
        sub     dx,dx
        div     bx
        add     ax,ax
        add     ax,ax
        add     ax,1970
        cmp     dx,365
        @if     ae
                sub     dx,365
                inc     ax
                cmp     dx,365
                @if     ae
                        sub     dx,365
                        inc     ax
                        cmp     dx,366
                        @if     ae
                                sub     dx,366
                                inc     ax
                        @endif
                @endif
        @endif
        mov     year,ax

        ; get day of year
        test    ax,3
        @if     z
                mov     bx,offset day_of_year2+2
        @else
                mov     bx,offset day_of_year+2
        @endif
        mov     month,0

        @begin
```

ワンラインNEWS
文化放送と日本電気は4月から首都圏で主婦向けのパソコン通信サービスをはじめる．パソコン通信網「PC-VAN」にフォーラムを設置し，まず1年間試験運用を行う．

【DPBの構造】

もう１つの構造体宣言は，DPBの構造を示すものです．ブロックデバイスのドライバでは，前記のようにBPBという形式でディスクの諸源をシステムに渡します．しかしこれだけだと，実際にアクセスするときに不便なので，BPBを解読し利用しやすい形にコンバートしたテーブルを使用しています．このテーブルがDPBであり，DOSのファンクションコール32Hでそのアドレスを取得することができます．

本来は，セクタサイズと予約セクタ数を含めた総セクタ数がわかれば，ディスクダンプは実行できます．これらはBPBには用意されているのですが，DPBには適切なものが用意されていません．ですからDPBに用意されている情報から計算してやる必要があります（デバイスドライバを直接呼んで，BPBを取得してもよいのですが……）．

なおこの構造は，DOS3. xのそれとなっています．DOS4以降では22～23ビットのセクタ番号がとれるので，wordで記述されている部分のいくつかがdwordになっていることでしょう．またDOS5では，int25/int26のアブソリュートリード・ライトの割り込みのパラメータの渡し方が変わっています．プログラムの中でも，セクタ数がwordで表現できるということでword精度しか取っていない部分がありますから，これらの部分を書き直さないとDOS4以降には対応できなくなります．ですから，今回のプログラムはDOS3. x限定ということになっています．

【画面出力マクロ】

次に，画面出力関係のマクロをいくつか定義してあります．以前のプログラムでは@stringと@putsでメッセージを出力していましたが，この程度のプログラムになるともう少し細かい画面表示が必要になります．そこで１文字出力とか固定文字出力などのマクロを定義することにしました．また，終了コードを伴って終了するexitマクロも定義しておきます．

【ワークエリア】

ワークエリアは基本的にはローカルで取ることにしてありますが，いくつかはグローバルで取っています．この手のプログラムでは，オプションの解読ルーチンがかなり手間かかります．Cで書かれたオリジナルのプログラムでは，以前に作ったライブラリを使い回していますが，それをアセンブラに書きなおすのはなかなかたいへんになってしまうので，今回はパーザーは簡略化してメインルーチンの中に仕込んでしまいました．オプション関係のワークエリアがグローバルになっているのはその名残です．あと，headerの実体が定義されています．

【バッファ】

テープとのやり取りは，ある程度まとまったブロックで行わないとパフォーマンスが上がりません．そこである程度の大きさのバッファを取ってやる必要があります．ここでは32KBのバッファを取っています．このイメージをEXEファイルでももっていると無駄なので，BSSセグメントに確保します．このためには，.data?という疑似命令を使用します．この疑似命令を使用するのは，たぶんこの連載でははじめてでしょう．これを通常のように_DATAセグメントに取ると，プログラムのサイズが32KB増えてしまいます．

【ライブラリ関数】

それから，ライブラリ関数が続きます．まず，Cのtime/ctimeに相当する関数が記述されています．DOS＋アセンブラで記述する場合には，このあたりはかなりめんどうになりますから，場合によっては流用することができるかもしれません．

まず，各月の初日が１月１日から通算して何日目になるかどうかを示すテーブルがあります．ほんとうはもっと最適化ができるでしょうが，閏年用に２つ用意してあります．また，オフセットをつけて月がそのままインデックスになるようにしてあります．

【関数time】

さて関数timeは，long変数へのポインタtime_varを受け取ります．最初にDOSのファンクションコールで時間を取得した後，これを０：０：０からの通算秒数に直します．mul命令を使用していますが，スピードはほとんど気にならないところでしか使用されないのでよいでしょう．１日の秒数は65536で入りきらないので，dword形式で処理する必要があります．これは変数に格納しておきます．渡されてくるのはテーブルのポインタですから，BXでインダイレクトアクセスします．

次に，1970/１/１からの通算日数を計算します．閏年であるかどうかは年（DOSで取得されるのは年そのもの）が４で割り切れるかどうかで判断できます．これを利用して，まず月の初日のオフセットをテーブルから取得します．それに日にちを加算します．それから年の処理に移るのですが，まず1970を引いてから365をかけます．閏年のあった年数は，その残余に１をたして４で割れば計算できるので，その

ワンラインNEWS
日本電気は九州日本電気の半導体工場に，次世代メモリである16MビットDRAM用の試作ライン導入し，16Mの需要本格化に即応できる体制を整える．

```
                inc     month
                inc     bx
                inc     bx

                cmp     dx,[bx]
        @until  b
        sub     dx,[bx-2]
        inc     dx
        mov     date,dx

        ; convert second to hour/minuto
        mov     ax,tsecond_of_day
        sub     dx,dx
        mov     bx,1800
        div     bx
        mov     hour,ax
        xchg    ax,dx
        sub     dx,dx
        mov     bx,30
        div     bx
        mov     minute,ax
        xchg    ax,dx
        add     ax,ax
        add     second,ax

        ; create time string
        mov     di,offset ctime_buffer
        push    ds
        pop     es

        ; first day name of week
        mov     ax,day_offset
        sub     ax,3
        mov     bx,7
        sub     dx,dx
        div     bx
        mov     si,dx
        add     si,si
        add     si,si
        add     si,offset day_name
        movsw
        movsw

        ; next, month name
        mov     si,month
        dec     si
        add     si,si
        add     si,si
        add     si,offset month_name
        movsw
        movsw

        ; date, time..
        set_value <date,' '>
        set_value <hour,':'>
        set_value <minute,':'>
        set_value <second,' '>

        ; year out
        mov     ax,year
        sub     dx,dx
        mov     bx,100
        div     bx
        push    dx
        set_value <ax,-1>
        pop     ax
        set_value <ax,0>

        ; and return
        mov     ax,offset ctime_buffer
        ret
endf

func    strcpy  <dst,src>,<si di>
        cld
        mov     si,src
        mov     di,dst
        push    ds
        pop     es
        @begin
                lodsb
                stosb
                or      al,al
        @until  z
        ret
endf

func    memcpy  <dst,src,cnt>,<si di>
        cld
        mov     si,src
        mov     di,dst
        push    ds
        pop     es
        mov     cx,cnt
        rep     movsb
        ret
endf

func    print_value     <v>
        mov     ax,v
        sub     dx,dx
        mov     cx,10
        cmp     ax,cx
        @if     ae
                div     cx
                push    dx
                print_value <ax>
                pop     ax
        @endif
        add     al,'0'
        @putc   <ax>
        ret
endf

func    absread <dr,cnt,start,address>
        mov     ax,dr
        mov     bx,address
        mov     cx,cnt
        mov     dx,start
        push    bp
        int     25h
        pop     ax
        pop     bp
        ret
endf

func    abswrite        <dr,cnt,start,address>
        mov     ax,dr
        mov     bx,address
        mov     cx,cnt
        mov     dx,start
        push    bp
        int     26h
        pop     ax
        pop     bp
        ret
endf

;---------------------------
; entry point

entry:
        @setup
        push    _argc
        push    _argv
        call    _main
exit:
        dos     <4ch,,,>

;---------------------------
; main routine
;

func    main <argc,argv>
        local   dpb_pointer:dword
        local   sector_factor
        local   sector_per_buffer
        local   tape_fd
        local   total_sector,1
        local   s

        ; clear option flag
        sub     ax,ax
        mov     rflag,ax
        mov     mflag,ax
        mov     fflag,ax

        ; parse option
        mov     dx,argv
        inc     dx
        inc     dx

        @begin
                mov     bx,dx
                mov     bx,[bx]
                cmp     byte ptr [bx],'-'
        @while  z
                inc     bx
                mov     al,[bx]
                inc     bx
                cmp     al,'m'
                @if     z
                        ; message option
                        mov     marg,bx
                        mov     mflag,1
                @else
                cmp     al,'r'
                @e_if   z
                        ; restore option
                        mov     rflag,1
                @else
                cmp     al,'f'
                @e_if   z
                        ; file name option
                        mov     farg,bx
                        mov     fflag,1
                @else
                        ; unknown option
                        @print  <'bad option:'>
                        dec     bx
                        @puts   bx

                        @exit <1>
                @endif

                ; adjust argc/argv
                dec     argc
                mov     cx,argc
                mov     bx,dx
                @begin
                        mov     ax,[bx+2]
                        mov     [bx],ax
                        inc     bx
                        inc     bx
                        dec     cx
                @until  z
        @repeat

        ; check argc
        cmp     argc,2
        @if     nz
                @print  <'ddump [-mmessage] [-fname] [-r] drive',0dh,0ah>
                @exit   <1>
        @endif

        ; set drive number
        mov     bx,argv
        inc     bx
        inc     bx
        mov     bx,[bx]
        mov     al,[bx]
        or      al,20h
```

ワンラインNEWS
米ヒューレット・パッカードは5,000ドルを切る新型WSを発売する。IBM社やシリコン・グラフィックス社も年内に1万ドルを切る低価格機を投入し，対抗する。

結果を加算します．その年数の分だけよけいに日が経過しているからです．それから日のオフセットが1日ある分を補正し，秒単位に直します．

このとき，前記のように1日の秒数は16bitで表現できないことから，その1/2の値をかけてさらに2倍します．シフトとローテイトを使用しても同じことですが，このほうがわかりやすいように思います．ローテイトがキャリーを通すかどうかについては，CPUによってかなりまちまちになっていますから，下手をするとまちがえてしまいます．86系ではRCRとCが入るとキャリーを通すタイプで，RORと入らないとキャリーを通しません．これはZ80などのザイログニーモニックとは逆です．86ではCはthrough the CarryのCと解釈され，Z80ではCircularのCと解釈されるので反対になるのです．

最後にこれらの日にち分の秒オフセットをワークエリアに加算し，そのアドレスをAXレジスタに入れてから復帰します．

【関数ctime】

関数ctimeは，経過秒を年月日，時分秒に分解し，それを文字列になおすものです．バッファと月名・曜日をテーブルとして用意しておきます．向こうでは年月日の並びは日・月・年である場合と月・日・年である場合が混在しています．ですから月は数値でなく文字で示してやることが安全策として取られるようです．ctimeもそのような仕様になっています．

ctimeの下請けに関数set_valueがあり，これはes:diからの領域に，引数に取った値をリーディング0を含む2桁の数値にして格納する，というものです．さらに引数に取ったセパレータを追加格納します．ただし，セパレータが-1である場合にはそのままにします．ポインタをレジスタ渡しし，しかもdiをアップデートするのですからかなり行儀の悪い関数ということになります．手続きは，単に10で割って商と余りを数値に変換するだけで，AAAなどのテクニックは使用していません．

ctimeでは，まず与えられた秒数をシフトして2で割ります．前記のように割り算の都合です．それから経過日数と1日の中の秒数/2に分解します．まず日数のほうを先に処理し，年数を求めます．4年単位での経過時間を出し，それから1年ごとのオフセットを経過日数から引いていきます．ここで得られた年は西暦そのものですので，やはり4で割ることによって閏年の判断ができます．これに基づいて日数オフセットのテーブルアドレスを取得し，ループを回して月と日の値を計算します．日についてはオフセット1を加算しなければなりません．

次に，秒の処理をします．これは簡単に割って商と余りを求めていくだけです．最後に秒の余りは2倍して元の経過秒数の下位ビットと加算します．

以上の処理で年月日等の値がもとまりましたから，次にこれを文字列に変換します．まず，以前取っておいた経過日数に1970/1/1のオフセットを差し引いてから7で割って余りを求めることにより，当日の曜日を計算します．アドレスを求めたら，後はrep movswを2回実行することで曜日をコピーします．曜日の後にスペースが入っているのはそのためです．同様に月名もコピーし，その次に日，そして時分秒を格納します．日の場合はセパレータはスペースで，時分などの場合にはコロンを指定します．最後に年を設定しなければいけませんが，これは2桁ではないので100で割って2回にわけて設定します．セパレーター1の特例は，このために利用します．最後にバッファのアドレスをAXに返して復帰します．

【strcpy/memcpy】

strcpy/memcpyはそれぞれ同様なCの関数の置き換えで，仕様頻度も高くないので簡単なインプリメントになっています．場合によっては，rep movswを利用するように改造する必要があるかもしれません．print_valueはディスクの容量を出力するためのもので，引数を10進表示します．簡単にするために，再帰していますが再帰の回数は5回程度ですので問題はないでしょう．

【absread/abswrite】

absread/abswriteはディスクのアブソリュートリード・ライトを実行するもので，前記のように今回は3.xにのみ対応した形式となっています．行っているのはレジスタに値を設定してint25/26を実行しているだけですが，すべてのレジスタが破壊されるということなのでBPについてもスタックに保存しています．またこれらのルーチンはretfで復帰するので，スタックに残ったフラグを取り除いてやらなければいけません．これらは，本来int25/26ではなくてfar callで呼び出すべきなのかもしれません．

【main】

次にエントリ部分の記述があり，それからmainの記述に移ります．mainでは，いくつかのローカル変数を定義してあります．dpb_pointerはDPBへのfarポインタなのでdwordとなっています．sector_factorは物理セクタ

ワンラインNEWS
東大阪の操作パネルメーカーのデジタルは工作機械などに簡単に組み込め，パソコンのデータを表示できるタッチパネル付きフラットパネルを開発した．

```
        sub     al,'a'
        inc     al
        mov     ah,0
        mov     drive,ax

        ; check drive range
        mov     ax,drive
        @mc
                cmp     ax,1
        @cond   b
                cmp     ax,26
        @cond   ae
        @if
                ; bad drive, output error message
                @print  <'Bad Drive:'>
                mov     dx,drive
                add     dl,'@'
                @putc   <dx>
                @crlf

                ; and exit with error code
                @exit   <1>
        @endif

        ; get dpb address in dx:bx
        push    ds
        dos     <32h,,,drive>
        mov     dx,ds
        pop     ds

        ; check drive accessibility
        cmp     al,0ffh
        @if     z
                @print  <'Bad Drive:'>
                mov     dx,drive
                add     dl,'@'
                @putc   <dx>
                @crlf

                @exit   <1>
        @endif

        ; save it
        mov     word ptr dpb_pointer,bx
        mov     word ptr dpb_pointer+2,dx

        ; set tape device name if need
        cmp     fflag,0
        @if     z
                ; tape name not assumed, assign it
                @string <'stape'>
                mov     farg,dx
        @endif

        ; set parameters
        les     bx,dpb_pointer
        mov     ax,es:[bx].byte_per_sector
        mov     cl,9
        shr     ax,cl
        mov     sector_factor,ax

        mov     ax,size buffer
        sub     dx,dx
        div     es:[bx].byte_per_sector
        mov     sector_per_buffer,ax

        cmp     rflag,0
        @if     z,far
                ; save operation
                open    farg,O_WRONLY
                mov     tape_fd,ax
                @if     c
                        ; open error
                        @print  <'can''t open :'>
                        @puts   farg
                        @crlf
                        @exit   <1>
                @endif

                ; set backup time
                time <offset header.htime>

                ; set hcomment if need
                cmp     mflag,0
                @if     nz
                        strcpy <offset header.htime,marg>
                @endif

                ; calculate sector number
                les     bx,dpb_pointer
                mov     ax,es:[bx].total_cluster
                mov     dl,es:[bx].sector_per_cluster
                mov     dh,0
                inc     dx
                mul     dx
                add     ax,es:[bx].top_sector

                ; set physical sector number to works
                mov     total_sector,ax
                mov     l,ax

                ; calculate logical(512b) sector number
                mul     sector_factor
                mov     word ptr header.count,ax
                mov     word ptr header.count+2,dx

                ; write header data
                memcpy <offset buffer, offset header, size header>
                write   tape_fd,<offset buffer>,512>

                mov     s,0
                @begin
                        ; check not written sectors
                        mov     ax,l
                        or      ax,ax
                @while  nz
                        ; calc ratio
                        dec     ax
                        mov     bx,100
                        mul     bx
                        div     total_sector

                        ; print it
                        @string <'99% left',0dh>
                        push    dx
                        mov     di,dx
                        push    ds
                        pop     es
                        set_value <ax,'%'>
                        pop     dx
                        @puts   dx

                        ; decide sector count to read
                        mov     ax,sector_per_buffer
                        cmp     l,ax
                        @if     be
                                mov     ax,l
                        @endif

                        mov     bx,drive
                        dec     bx
                        push    ax
                        absread <bx,ax,s,offset buffer>
                        write   tape_fd,<offset buffer>,<size buffer>
                        pop     ax

                        add     s,ax
                        sub     l,ax
                @repeat
                @print  <'all done    ',0dh,0ah>
        @else
                ; restore operation
                open    farg,O_RDONLY
                mov     tape_fd,ax
                @if     c
                        ; open error
                        @print  <'can''t open :'>
                        @puts   farg
                        @crlf
                        @exit   <1>
                @endif

                ; read header data
                read    tape_fd,<offset buffer>,512
                memcpy <offset header,offset buffer,size header>

                ; print out header data
                @print  <'data backuped '>
                ctime <offset header.htime>
                @puts   ax
                @print  <', '>
                mov     ax,word ptr header.count
                mov     dx,word ptr header.count+2
                mov     cx,11
                @begin
                        shr     dx,1
                        rcr     ax,1
                        dec     cx
                @until  z
                print_value <ax>
                @print  <'MB to drive '>
                mov     ax,drive
                add     al,'A'
                dec     ax
                @putc   <ax>
                @print  <', "'>
                @puts   <offset header.hcomment>
                @print  <'"',0dh,0ah>

                ; confirm
                @print  <'restore, OK?'>
                dos     <1>
                push    ax
                @crlf
                pop     ax
                or      al,20h
                cmp     al,'y'
                @if     nz
                        ; abort
                        @print  <'aborted',0dh,0ah>
                        @exit   <2>
                @endif

                ; calculate physical sector count
                mov     s,0
                mov     ax,word ptr header.count
                mov     dx,word ptr header.count+2
                mov     cx,sector_factor
                dec     cx
                add     ax,cx
                adc     dx,0
                div     sector_factor
                mov     l,ax
                mov     total_sector,ax

                @begin
                        mov     ax,l
                        or      ax,ax
                @while  nz
                        ; calc ratio
                        dec     ax
                        mov     bx,100
                        mul     bx
                        div     total_sector

                        ; print it
                        @string <'99% left',0dh>
                        push    dx
                        mov     di,dx
                        push    ds
                        pop     es
                        set_value <ax,'%'>
```

ワンラインNEWS

京セラは主力事業の半導体パッケージ部門で，92年度はWS向けに重点投資していく方針．同社は92年度から国内メーカーが相次いでWSの発売に踏み切ると見ている．

```
                pop     dx
                @puts   dx

                ; decide sector count to read
                read    tape_fd,<offset buffer>,<size buffer>

                mov     bx,drive
                dec     bx
                mov     ax,sector_per_buffer
                cmp     l,ax
                @if     be
                        mov     ax,l
                @endif
                push    ax
                abswrite <bx,ax,s,offset buffer>
                pop     ax

                add     s,ax
                sub     l,ax
        @repeat
        @print  <'all done        ',0dh,0ah>
    @endif
    @exit <0>
endf

end     entry
```

がいくつの論理セクタ(512バイトサイズ)に相当するかを示し，sector_per_bufferはバッファ(ここでは8000hバイト)が何物理セクタに対応するかを示します．tape_fdはテープアクセス用のデスクリプタであり，total_sectorは総セクタ数を示します．l は残セクタ数であり，s は次に読み書きするセクタ番号を示します．

プログラムの動作

プログラムはまずオプションのフラグをクリアした後，オプションの解析に移ります．argvを取得して，その引数が－で始まっていれば，次の文字によって処理を分岐します．引数を伴うオプションである場合には，そのアドレスをワークエリアに格納します．不正なオプションが与えられれば，即座にメッセージを出力して終了します．正しいオプションだった場合には，解釈を終えた後argcをデクリメントし，argvの内容から解釈したオプションを取り除きます．

次にドライブ名が引数として残っているかどうかを調べ，argcが不正な値であればやはり中断します．正しければ引数をドライブ名と解釈し，そのレンジチェックをします．いずれも問題なければDPBのアドレスを取得します．ここで不正なドライブであればALにFFHが返ってくるので，それをチェックします．

それから，f オプションが与えられていない場合にはデフォルトのテープデバイス名をセットし，DPBを解釈して sector_factorなどの計算をします．そして r オプションのフラグを調べて，セーブするかリストアするかに分岐します．

セーブする場合には，まずテープデバイスを書き込みモードで開いた後，ヘッダに時間と必要ならコメントの内容を設定します．次に総セクタ数の計算をします．セクタ数は「クラスタ数＊クラスタあたりのセクタ数＋データ領域の最初のセクタ番号」で計算できます．クラスタあたりのセクタ数は，DPBに入っている値は１を引いた値なので調整しないといけません．ここで計算された値は16ビットを超えないので，それらをwordのワークエリアに格納します．次に，512バイトセクタでの論理セクタ数に直しこれは32ビットの値としてヘッダ部分に格納します．それからセクタをバッファ領域にコピーし，512バイトだけまずデバイスに書き出します．

それから残りセクタがある間，ドライブから読み出し，デバイスに書き込みを繰り返します．ループの最初で，残っているパーセンテージを出力します．これは，「残りのセクタ数＊100／全セクタ数」で計算します．ただし，残りのセクタ数は１を引いて，最初のときでも100%にならないようにします．計算した値の10進化はset_valueを流用して，値の設定された文字列を上書きしてしまっています．これを毎回出力するわけです．読み込みをするセクタ数は，バッファを満たすセクタ数または残りのセクタ数の小さいほうですが，デバイスに書き出すときにはバッファ全部を書き出してしまっています．ですから，最後にはゴミがついている可能性があります．

読み込みの場合は，ほぼ逆の動作をします．やはりデバイスをオープンしてから１セクタ分だけバッファに読み込み，これをヘッダにコピーして内容を出力します．それから処理を継続するかどうかを聞いてきて，y が入力されなければ処理を中断します．継続された場合にはヘッダの論理セクタ数から処理すべき物理セクタ数を計算し，以後同様に処理をします．

プログラムの作成

このプログラムは，通常どおりtasmでアセンブルし，tlinkでリンクすることによって実行可能なexeファイルを作成できます．もちろん，実行に先立ってxscsi. sysとxtape. sysがインストールされていなければなりません．バックアップする際にはmtコマンドで巻き戻してからこのプログラムを実行し，最後に再びmtコマンドで巻き戻しするのがよいでしょう．

バックアップの場合は問題ないと思いますが，リストアの場合はキャッシュなどを外して，かつ終了した後リセットすることを心掛けたほうがよいでしょう．

ワンラインNEWS
米デジタル・イクイップメントは近くパソコンの通信販売制度を導入する．パソコンの通信販売は米デル・コンピュータ社が採用し，成功している．

情報処理技術者試験

第2種合格！トレーニング

第12回　問題と解答 —— 午後の部

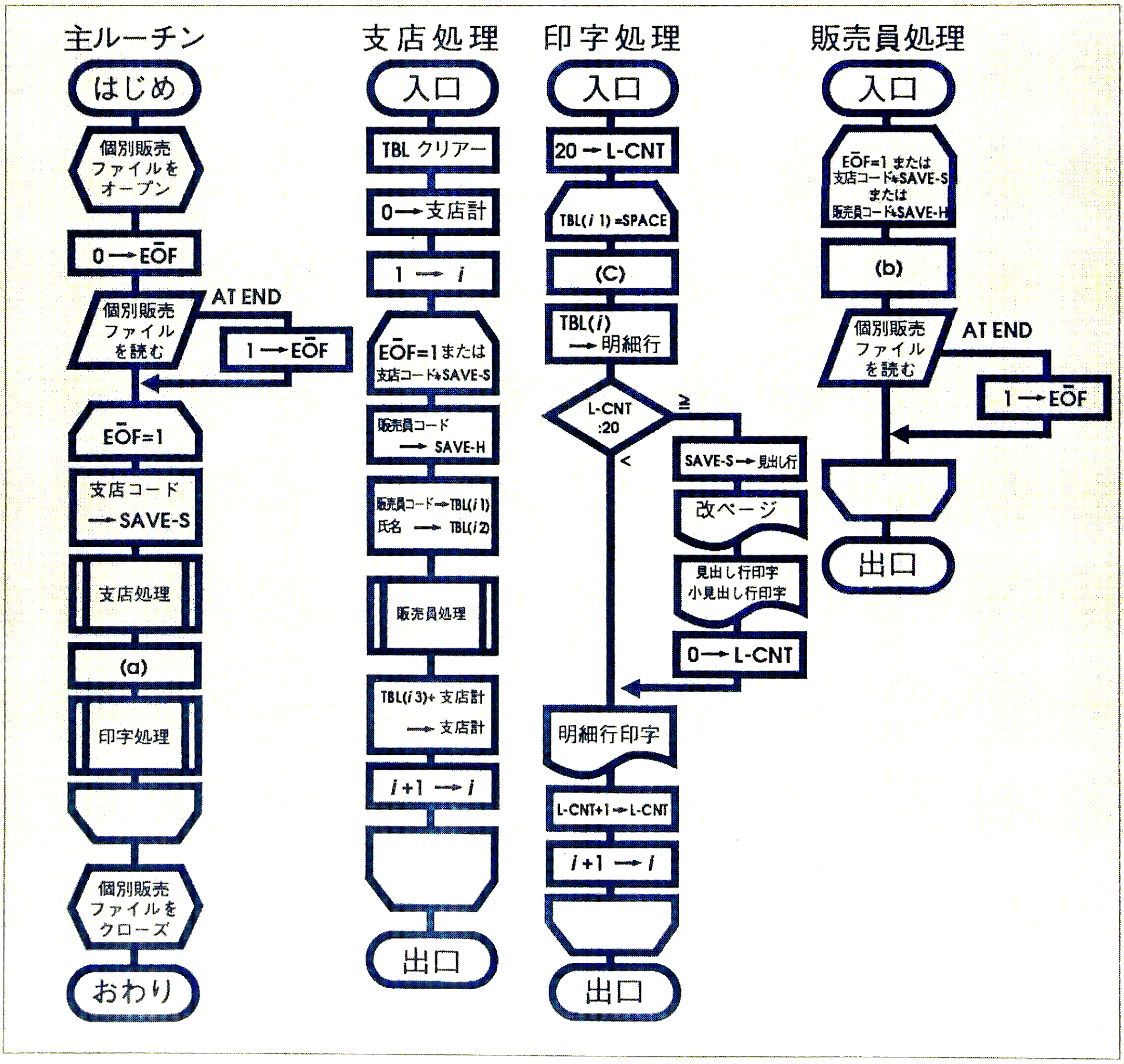
静岡情報ビジネス専門学校

前月につづき，試験前の対策として，午後の部の出題範囲から，おさえておきたい問題とその解答を掲載します．

問　1

図1の流れ図中の（a）～（c）に入れるべき適当な字句を，図にならって記入せよ．

●問1－図1



[流れ図の説明]

個別販売ファイルから販売員ごとの合計金額を求め，支店の売上金額に占める割合を含む売上成績表リストを作成する．

（1）個別販売ファイルは，支店コード，販売員コードの昇順に並んでいる（図2）．

なお，同一支店コード，販売員コードが複数存在する．

（2）1ページには明細20行とする．もしくは，支店が変わるごとにページを改めること．

（3）売上成績表リストは図3のとおりである．

（4）作業域として，配列TBLをおく（図4）．

解　答

（a）1 → i

（b）TBL（i 3）＋ 金額 → TBL（i 3）

（c）TBL（i 3）／支店計 × 100 → TBL（i 4）

●問1－図2

支店ｺｰﾄﾞ	販売員ｺｰﾄﾞ	氏　名	金　額

●問1－図3

```
                     売 上 成 績 表              支店ｺｰﾄﾞXX

販売員ｺｰﾄﾞ     氏    名        金     額        (%)

   XX       XXX  XXX       X,XXX,XXX       XX.X
   XX       XXXX  XXX        XXX,XXX        X.X
   XX       XXX  XXXX      X,XXX,XXX       XX.X
   :            :              :             :
```

●問1－図4

01	ｴﾃｨ ﾏﾋｨｰ	1,260,000	10.5
02	ｼｮﾝ ﾗﾝﾎﾞｰ	750,000	6.3

問　2

リスト1のCOBOLプログラム中の［　　］を埋めて，プログラムを完成させよ．

[プログラムの説明]

販売台数売上ファイルを読んで，販売台数一覧表を印字するプログラムである．

（1）販売台数ファイルのレコード様式は，図1の通りである．

●問2－図1

車コード	車　名	販売台数
10桁	20桁	5桁

●問2－図2

```
              ****   ｳﾘｱｹﾞﾀﾞｲｽｳ     ｲﾁﾗﾝﾋｮｳ   ****              PAGE  1
ｼｬﾒｲ            ﾀﾞｲｽｳ                                                 ﾋｮｳｶ
ﾄﾖﾀﾞ  ｱﾗｿ        250   **********                                      E
ｹﾞｯｻﾝ ﾏｰｼ       1005   ****************************************        D
```

①データはチェック済みでエラーはないものとする．

（2）印刷様式

①1ページに明細25行とし，ページ制御を行う．

②“＊”は販売台数25につき1つ印字するものとし，端数は切上げとする．販売台数が2500以上は，100個印字するものとする．

③ヒョウカ欄は

“＊”が40未満の場合　　　“E”

“＊”が40以上60未満の場合　“D”

“＊”が60以上80未満の場合　“C”

“＊”が80以上100未満の場合“B”

“＊”が100の場合　　　　“A”

を印字する．

解　答

①MOVE　ZERO　TO　W-LINE　W-PAGE

または

INITIALIZE　W-LINE　W-PAGE

②COMPUTE　S-SU　＝　(C-DAISU － 1)/25＋1

③MOVE SPACE TO M-TBL

または

INITIALIZE　M-TBL

④COMPUTE　I　＝　I　－　1

⑤COMPUTE　W-LINE　＝　W-LINE　－　1

●リスト1

```
 DATA                    DIVISION.
 FILE                    SECTION.
 FD  CAR-FILE.
 01  CAR-REC.
     02  C-NO        PIC X(10).
     02  C-NAME      PIC X(20).
     02  C-DAISU     PIC 9(05).
*
 FD  O-FILE.
 01  O-REC           PIC X(132).
*
 WORKING-STORAGE         SECTION.
 01  I               PIC 9(03).
 01  S-SU            PIC 9(05).
 01  W-LINE          PIC 9(02).
 01  W-PAGE          PIC 9(02).
 01  EOF             PIC 9(01)  VALUE  ZERO.
*
 01  OOMIDASHI.
     02              PIC X(50)  VALUE  SPACE.
     02              PIC X(37)
             VALUE "****   ｳﾘｱｹﾞﾀﾞｲｽｳ     ｲﾁﾗﾝﾋｮｳ     ****".
     02              PIC X(32)  VALUE  SPACE.
     02              PIC X(05)  VALUE  "PAGE ".
     02    O-PAGE    PIC Z9.
*
 01  KOMIDASI.
     02              PIC X(02)  VALUE  SPACE.
     02              PIC X(04)  VALUE  "ｼｬﾒｲ".
     02              PIC X(10)  VALUE  SPACE.
     02              PIC X(05)  VALUE  "ﾀﾞｲｽｳ".
     02              PIC X(100) VALUE  SPACE.
     02              PIC X(05)  VALUE  "ﾋｮｳｶ".
*
 01  MEISAI.
     02              PIC X(02)  VALUE  SPACE.
     02    M-NAME    PIC X(10).
     02              PIC X(01)  VALUE  SPACE.
     02    M-DAISU   PIC ZZ,ZZ9.
     02              PIC X(01)  VALUE  SPACE.
     02    M-TBL.
           03  HOSI  PIC X(01)  OCCURS 100.
     02              PIC X(05)  VALUE  SPACE.
     02    M-HYOKA   PIC X(01).
*
 PROCEDURE               DIVISION.
 HAJIME.
     OPEN  INPUT CAR-FILE  OUTPUT  O-FILE
     [            ①            ]
     READ  CAR-FILE  AT END  MOVE 1  TO  EOF  END-READ
     PERFORM  UNTIL  EOF  =  1
         IF  W-LINE  =  ZERO
             PERFORM KAIPAGE
         END-IF
         MOVE     C-NAME  TO  M-NAME
         MOVE     C-DAISU TO  M-DAISU
     [            ②            ]
         IF  S-SU  >  100
             MOVE  100    TO  S-SU
         END-IF
     [            ③            ]
         PERFORM  VARYING  I  FROM  1  BY  1  UNTIL  I  >  S-SU
             MOVE  "*"    TO  HOSI(I)
         END-PERFORM
     [            ④            ]
        EVALUATE  TRUE
            WHEN  I  <   40
                  MOVE  "E"  TO  M-HYOKA
            WHEN  I  >=  40  AND  I  <  60
                  MOVE  "D"  TO  M-HYOKA
            WHEN  I  >=  60  AND  I  <  80
                  MOVE  "C"  TO  M-HYOKA
            WHEN  I  >=  80  AND  I  <  100
                  MOVE  "B"  TO  M-HYOKA
            WHEN  I  =   100
                  MOVE  "A"  TO  M-HYOKA
        END-EVALUATE
        WRITE    O-REC  FROM  MEISAI  AFTER  2
     [            ⑤            ]
         READ  CAR-FILE  AT END  MOVE 1  TO  EOF  END-READ
     END-PERFORM
     CLOSE  CAR-FILE  O-FILE
     STOP   RUN.
 KAIPAGE.
     COMPUTE   W-PAGE   =     W-PAGE     +   1
     MOVE      W-PAGE   TO    O-PAGE
     WRITE     O-REC    FROM  OOMIDASI   AFTER  PAGE
     WRITE     O-REC    FROM  KOMIDASI   AFTER  2
     MOVE      25       TO    W-LINE.
```

問　3

リスト2のCOBOLプログラム中の［　　］を埋めて，プログラムを完成せよ．

［プログラムの説明］

旧商品マスタファイルと修正ファイルをもとに新マスタファイルを作成する．旧商品マスタファイル，修正ファイルは，順編成ファイルであり商品コードの昇順に分類済みである．また，同一商品コードに対する修正レコードは1件のみである．

（1）旧商品マスタファイルのレコード様式は，図1の通りである．

（2）新商品マスタファイルのレコード様式は，図2の通りである．

（3）修正ファイルのレコード様式は，

図３の通りである。

（４）修正ファイルの区分は下記の通りである。

1……品名のみ修正
2……数量のみ修正
3……削除
4……追加

（５）修正ファイルの商品コードが旧商品マスタファイルに存在し，区分が1～3以外の場合，また修正ファイルの商品コードが旧商品マスタファイルに存在せず，区分が4以外の場合にエラーデータとし，警告のためプリンタに出力する。エラーデータには，“＊”を付加する。また，削除データもプリンタに出力する。

なお，見出しや改ページは考慮しなくともよい。

●問３－図１

商品ｺｰﾄﾞ	商　品　名	数　量
３けた	１５けた	６けた

●問３－図２

商品ｺｰﾄﾞ	商　品　名	数　量
３けた	１５けた	６けた

●問３－図３

商品ｺｰﾄﾞ	区分	商　品　名	数　量
３けた	１けた	１５けた	６けた

●問３－図４

```
          ＊＊＊   削除  エラーデータ   一覧表    ＊＊＊
   商品コード  区分    商    品    名      数   量       エラー
     001        3     ｴﾝﾋﾟﾂ                 1,253
     003        8     ｹｼｺﾞﾑ                   521          ＊
      ：        ：      ：                      ：           ：
```

●リスト２

```
DATA DIVISION.
FILE SECTION.
FD  OMST-FILE.
01  OMST-REC.
    02  O-CODE     PIC X(03).
    02  O-NAME     PIC X(15).
    02  O-SU       PIC 9(06).
FD  NMST-FILE.
01  NMST-REC.
    02  N-CODE     PIC X(03).
    02  N-NAME     PIC X(15).
    02  N-SU       PIC 9(06).
FD  TRN-FILE.
01  TRN-REC.
    02  T-CODE     PIC X(03).
    02  T-KUBUN    PIV X(01).
    02  T-NAME     PIC X(15).
    02  T-SU       PIC 9(06).
FD  OUT-FILE.
01  OUT-REC           PIC X(132).
WORKING-STORAGE SECTION.
01  MEISAI.
    02             PIC X(20) VALUE SPACE.
    02  M-CODE     PIC X(03).
    02             PIC X(10) VALUE SPACE.
    02  M-KUBUN    PIC X(01).
    02             PIC X(10) VALUE SPACE.
    02  M-NAME     PIC X(15).
    02             PIC X(10) VALUE SPACE.
    02  M-SU       PIC ZZZ,ZZ9.
    02             PIC X(10) VALUE SPACE.
    02  M-ERR      PIC X(01).
PROCEDURE DIVISION.
INIT-RTN.
    OPEN INPUT  OMST-FILE TRN-FILE
         OUTPUT NMST-FILE OUT-FILE
  ┌──────────────────────────────┐
  │             (1)              │
  └──────────────────────────────┘
MAIN-RTN.
    PERFORM UNTIL O-CODE = HIGH-VALUE AND T-CODE = HIGH-VALUE
        EVALUATE TRUE
            WHEN O-CODE > T-CODE
                EVALUATE T-KUBUN
  ┌──────────────────────────────┐
  │             (2)              │
  └──────────────────────────────┘
                        MOVE T-CODE TO N-CODE
                        MOVE T-NAME TO N-NAME
                        MOVE T-SU   TO N-SU
                        WRITE NMST-REC
                    WHEN OTHER
                        PERFORM ERR-RTN
                END-EVALUATE
  ┌──────────────────────────────┐
  │             (3)              │
  └──────────────────────────────┘
  ┌──────────────────────────────┐
  │             (4)              │
  └──────────────────────────────┘
                MOVE O-CODE TO N-CODE
                MOVE O-NAME TO N-NAME
                MOVE O-SU   TO N-SU
                WRITE NMST-REC
  ┌──────────────────────────────┐
  │             (5)              │
  └──────────────────────────────┘
            WHEN OTHER
                EVALUATE T-KUBUN
                    WHEN "1"
                        MOVE O-CODE TO N-CODE
                        MOVE T-NAME TO N-NAME
                        MOVE O-SU   TO N-SU
                        WRITE NMST-REC
                    WHEN "2"
                        MOVE O-CODE TO N-CODE
                        MOVE O-NAME TO N-NAME
                        MOVE T-SU   TO N-SU
                        WRITE NMST-REC
                    WHEN "3"
                        MOVE O-CODE  TO M-CODE
                        MOVE T-KUBUN TO M-KUBUN
                        MOVE O-NAME  TO M-NAME
                        MOVE O-SU    TO M-SU
  ┌──────────────────────────────┐
  │             (6)              │
  └──────────────────────────────┘
                        WRITE OUT-REC FROM MEISAI AFTER 2
                    WHEN OTHER
  ┌──────────────────────────────┐
  │             (7)              │
  └──────────────────────────────┘
                END-EVALUATE
```

```
        ┌──────────(1)──────────┐
            END-EVALUATE
        END-PERFORM.
    END-RTN.
        CLOSE OMST-FILE NMST-FILE TRN-FILE  OUT-FILE
        STOP RUN.
    M-READ.
        READ OMST-FILE AT END MOVE HIGH-VALUE TO O-CODE
        END-READ.
    T-READ.
        READ TRN-FILE  AT END MOVE HIGH-VALUE TO T-CODE
        END-READ.
    ERR-RTN.
        MOVE "*" TO M-ERR
        MOVE T-CODE TO M-CODE
        MOVE T-KUBUN TO M-KUBUN
        MOVE T-NAME TO M-NAME
        MOVE T-SU TO M-SU
        WRITE OUT-REC FROM MEISAI AFTER 2.
```

エラーデータ，削除データは新マスタファイルに出力しない．

解　答

（1）PERFORM M-READ THRU T-READ
（2）WHEN "4"
（3）PERFORM T-READ
（4）WHEN O-CODE < T-CODE
（5）PERFORM M-READ
（6）MOVE SPACE TO M-ERR
（7）PERFORM ERR-RTN

●リスト3

```
  BINDEC  START
          PUSH      0,GR0                 レジスタ退避
          PUSH      0,GR1
          PUSH      0,GR2
          PUSH      0,GR3
          PUSH      0,GR4

          LEA       GR1,0
          ST        GR0,WORK1
   LOOP1  ST        GR0,WORK
  ┌───────────①───────────┐               除算処理
          JMI       NEXT
          ST        GR0,WORK              剰余算出
          JMP       LOOP1

  NEXT    LD        GR0,WORK
  ┌───────────②───────────┐               文字コード変換
          ST        GR0,BUF,GR1
          LEA       GR1,1,GR1
          LD        GR0,WORK1
          CALL      DIV10                 10で除算
          ST        GR0,WORK1
          CPA       GR0,0                 除算結果の判定
  ┌───────────③───────────┐
          JPZ       LOOP1

   LOOP2  ┌─────────④─────────┐
          LEA       GR2,0
   LOOP3  CPA       GR2,0,GR1
  ┌───────────⑤───────────┐
          LD        GR3,BUF,GR2           バッファデータ交換
          LD        GR4,BUF,GR1
          ST        GR4,BUF,GR2
          ST        GR3,BUF,GR1
  ┌───────────⑥───────────┐
  ┌───────────⑦───────────┐
          JMP       LOOP3

  SHURYO  POP       GR4
          POP       GR3
          POP       GR2
          POP       GR1
          POP       GR0

          EXIT
  WORK    DS        1
  WORK1   DS        1
  BUF     DS        10
  TEN     DC        10
  CHAR0   DC        #0030
          END
```

問　4

次のCASLプログラムの説明，およびプログラムを読んで，設問に答えよ．

［プログラムの説明］

（1） 2進数を外部出力装置に出力できる形の5桁の10進数に変換するプログラムである．

（2）2進数値はGR0に入れて，このプログラムに渡される．

（3） 変換データを格納する領域は最大10語とする．

（4）サブルーチンDIV10は，GR0を10で割り，商をGR0に設定する．

設　問

プログラム（リスト3）中の空所に適当な命令を設定して，プログラムを完成せよ．なお，ラベル欄は必要でない限り空欄とする．

解　答

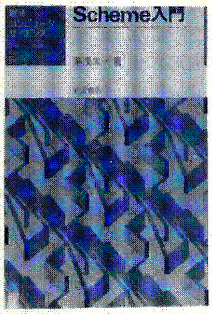
① SUB　GR0,TEN
② ADD　GR0,CHAR0
③ JZE　LOOP2
④ LEA　GR1,-1,GR1
⑤ JPZ　SHURYO
⑥ LEA　GR1,-1,GR1
⑦ LEA　GR2,1,GR2

Book Review

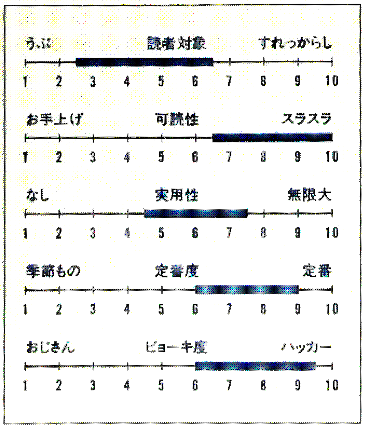
■中村正三郎

■Scheme入門

湯浅太一著
岩波書店
3,200円



■評価チャート



Lispの信頼すべき導師によるもうひとつの標準——Scheme

現存するプログラミング言語の中でFORTRANに匹敵する古い歴史を持ちながら，いまだに現代性を失っていないのがLispである．現代性を失うどころか，その柔軟性，拡張性，高い記述能力，先進性によって時代を先取りし，ハッカーを魅了し続けてきた．

Lispはインプリメントが比較的容易であり，アイデアを盛り込むのに適した言語であったがために，これまで数多くの方言が作られた．MITで大昔に開発されたMacLispの流れを汲むLispについては，80年代になって統一の機運が高まり，現在Common Lispとして仕様がまとまっている．

そしてもうひとつの標準として注目されているLispがSchemeである．これはCommon Lisp制定の中心人物スチール氏と後述のサスマン氏らがMITで開発したもので，Common Lispにも影響を与えたLispである．Common Lispに比べ非常に簡素な言語仕様であり，それでいて強力（特に継続をユーザレベルで操作できる点は特筆に値する）．評者の周辺ではCommon Lispよりはるかに人気の高いLisp仕様である．

本書の著者，湯浅氏は京都大学数理解析研究所の萩谷昌己氏とともに，Common Lispの処理系を世界に先駆けて開発し，公開した実績がある．このLispはKCL（Kyoto Common Lisp）の名で世界的に著名なCommon Lisp処理系となった．

また湯浅氏には同じ岩波のシリーズに，萩谷氏との共著による『Common Lisp入門』（岩波書店），そしてそのドリルの『Common Lispドリル』（岩波書店）がある．この2つは英訳もされている．

つまり，湯浅氏は，手がけた処理系，著書の実績からしてもLisp世界の信頼すべき導師といえる．その人の手によるScheme入門である．安心してお買いになればよい．

本書は，先の『Common Lisp入門』と『Common Lispドリル』を足して2で割ったような構成である．記述は簡潔で要を得ているが，まったく初めてLispに接する読者にはその淡泊さがとっつきにくいかもしれない．それを補う意味でも，各章の最後にまとめと演習問題があるのは，とてもよい．さらによいのは演習問題の解答が載っていることだ．初学者にはたいへんに助かるだろう．

内容は基本から始まり，入門といいつつも，クロージャ，バッククォートを使ったマクロ，数式処理，継続を使ったCatch ＆ Throwやコルーチンの実現など，高度なものまで含んでいる．演習問題を解きながら読み進めれば，Lispの実力はかなりのものとなるだろう．

編集レイアウト上のミスというべきか，演習問題と解答部分の右ページのヘッダ部分には，その章，最後の節のタイトルが延々と印刷されている．フォーマッタの設定が原因と思われるのだが，ここはやはり，「第5章演習問題」「第5章解答」などとしてほしかった．第5章の当該部分，約10ページにわたって「ごみ集め」となっており，ぱらぱらめくっていたとき，こんなにページを割いてガベージコレクションの説明をしているのか早とちりをしてしまったのだ．

本書に対応したScheme処理系として，湯浅氏のグループが開発したTUT-Schemeが別売になっている．日本語が使えない（動作が保証されていない）などいくつか残念な点はあるが，Emacsに似たエディタを内蔵し，PC-9800ならグラフィックスも使える．カタログ通り高速な処理系である．

別売ディスクには，本の例題プログラムや演習問題の解答プログラムを入れてほしかった．入力の手間が省けるし，初学者は括弧の対応をひとつ間違えただけでどうしていいか立ち往生す

ワンラインNEWS
日本電気は財団法人保安電子通信技術協会などと共同で，警察署関連の業務用ソフトウェア開発会社「公共情報管理システム」を設立した．

ることが多く，すぐ動かせるようプログラムを収めてほしかったのだ．そして，できればTUT-Schemeのソースを公開してほしかった．ソースを読むことによってLisp内部へのより深い理解が得られ，これほど学習に適したものはないからである．

なお，さまざまなScheme処理系が，日経MIXなどのネットワークにフリーソフトウェアとしてアップしてあるので，それを用いるのも一案である．ソース付きのものも多いので，Lispの内部を探る勉強にもなるだろう．ただし，各処理系で少しずつ仕様が違うことが多く，動かない例もあるかもしれない．

本書を卒業したら，ぜひ，アベルソン，サスマン著，元吉文男訳『プログラムの構造と実行(上下)』(マグロウヒル)に挑戦してください．Schemeを用いたプログラミングの名著です．

UNIX，Windows関連の訳本が目白押し

古田島義和（神田・書泉グランデ）

昨年暮れからUNIX，WindowsのOS関連図書の訳本や書き下ろしが本格的にはじまり，とくに92年は，訳本が多くなりそうです．既刊本としては，富士ソフトウエア「UNIXガイド」，Bantam「Peter Norton Guide to UNIX」，HBJ出版「UNIX System V ユーザ活用ガイド」，Howard. W. Sons「Exploring the UNIX System」，トッパン「UNIX/Cソフトウェアエンジニアリング」，Prentice Hall「Software Engineering with UNIX/C Environment」，日経BP社「リアルタイムUNIXシステム」，Kluwer Academic「Real Time UNIX Systems」．

そして注目のシリーズは，表紙が動物のイラストでお馴染みのO'ReilyのUNIXシリーズです．大きく分けて6分野あり，UNIX and C programming, UNIX System Administration, UNIX Basics, UNIX Text processing, UNIX Communication, UNIX Securityで，現在出版点数も30点以上と内容ともにすばらしいものがあります．そして昨年12月に同社の社長であるTim O'Reily氏が小社を訪れて「今後もUNIXを中心とした出版活動を続けていく」と話され，同時に日本での訳本が表紙も同じ装丁で，売れ行き好調であることに大満足して帰られました．すでに訳本は，啓学出版，アスキーから5点出版され，いずれもベストセラーになっています．そして今回もD. Dougberty氏の「Sed&Awk」が「sed&awkプログラミング」，T. O'Reily, G. Todino共著の「Managing UUCP and Usenet」が「UUCPシステム管理」としてアスキーより出版され，ベスト10の第1位と3位に登場，洋書のベスト10でも1位にD. Cameron氏の「Learning GNU Emacs」と8位A. Frisch「Essentical System Administration」が入りO'Reilyが両方で上位を独占，今年の台風の目といったところです．

一方のWindows分野では，Visual Basic関係図書が早くも登場，ビー・エヌ・エヌ「Visual Basicプログラミング技法」で原書はBantam「Visual Basic programming with Windows Application」，書き下ろしでは，ビレッジセンターから「98シリーズで使うVisual Basic入門」が出版され，2冊とも好調な売れ行きを続けています．本書はWindows3.0における新しいプログラミング環境で，WindowsのGUIを使用したアプリケーションの開発にもっとも操作性の高いツールを提供するソフトの解説書として注目を集めています．先月紹介したMicrosoft Press「Microsoft Visual Basic Workshop」が発売以来評判通りの売り上げを見せています．

言語では，相変わらずC言語が好調のようです．ナツメ社「間違いだらけのCプログラミング」，工学図書「BIOSとCプログラミング」といったものが健闘しています．あとは，エーアイ出版のGNU AWKプログラミング言語を使った「入門JGAWK」が7位で，さすがAWKという感じです．JGAWKはMS-DOS3.3C上で動かすことができるソフトでGNU AWKに興味のあるユーザーには必読書でしょう．最後にModula言語を勉強している方には，Prentice HallからS. P. Harbison氏の「Modula-3」が近日入荷します．すでに同社のG. Nelson「Systems programming with Modula-3」が発売と同時に売り切れ，人気(?)の高さがわかります．

■和書&洋書ベスト10　（書泉グランデ調べ：1991年12月1日～12月31日）

1	sed&awkプログラミング	アスキー
2	スーパーエンジニアへの道	共立出版
3	UUCPシステム管理	アスキー
4	UNIXオブザーバー92.1 No.2	スペック
5	カッコウはコンピュータに卵を産む(上)	草思社
6	電子立国 日本の自叙伝(中)	日本放送出版協会
7	入門JGAWK	エーアイ出版
8	カッコウはコンピュータに卵を産む(下)	草思社
9	LATEX美文書作成入門	技術評論社
10	NIFTY-Serve '91年版	星雲社

1	Learning GNU Emacs	O'Reily
2	Eiffel:The Language	Prentice Hall
3	C++ Programming Language 2/E	Addison Wesley
4	Standard C Library	Prentice Hall
5	Undocumented DOS Ver.5	Addison Wesley
6	Structure&Interpretation of Computer programming	M.I.T
7	Compilers(paper)	Addison Wesley
8	Essentical System Administration	O'Reily
9	MS-DOS programming Reference 2/E	Microsoft press
10	Turbo pascal for Windows 3.0 programming	Bantam

★ ★ ★ 佐々木哲の ★ ★ ★ U.S. プロダクツウォッチ

★DOS/Vは誰のもの？ 早くしないと……

AXとは違う形態ながら，日本語化したIBM互換機の規格であるDOS/V．どうもまだまだ混乱が続いていますね．各社から供給されるはずのDOS/Vも，未だにIBMブランドのものしかありません．いったいどうなっているのでしょうか？　どのメーカーも自分のブランドのDOS/Vをちゃんと用意して，早く売り出したいところだと思います．どこで供給が止まっているのやら……．

DOS/VはIBM主導で作られた規格です．それでも，これがもっと広まるためにはOADG参加の，あるいは在野のメーカー，ソフトハウスの努力が期待されるところです．ところがどうも，IBMやOADGに過大な期待をしているところがあります．

また互換性に関する議論もずれている気がするのですがどうでしょうか？　国家機関的な互換性の承認もいいのですが，やはり互換性というのは市場で揉まれて育つものでしょう．ハードウェアメーカーだけでなく，ソフトウェアメーカーもハードウェアをガチガチに触る時代ではなくなっています．お互いに常識を得て互換性を保ち，その上で独創性を出していく．そうでないとDOS/VもAXの二の舞でしょう．

最近，台湾や韓国などの自国語DOSの状況を知る機会がありました．ずっとIBM互換機だけで育ってきた彼らと歴史の長さが違うのは当たり前ですし，日本の市場が保守的で小さいのもわかりますが，活気が異なります．たとえば，表示ドライバに完全にソース公開のパブリックドメインのものまであるのです．規格は後から付いて行くぐらいの活力がないといけません．

すでに硬化がはじまった既存の国産機の市場とは違い，DOS/Vはまだ新しい市場です．ソフトハウスや周辺機器メーカーには既成の概念にとらわれない大きなチャンスがあると思います．DOS/Vでしかできないことはいろいろあります．たとえば，IBM互換機用のゲームソフトをいち早く移植したアスキーなどよい例といえます．

★ビデオ専用のローカルバス

さて，MS-Windowsのドライバの統一的な日本語化の手法も開発され，日本でも一気に普及が期待されるところです．今ではS3などのWindows用アクセラレータ付きVGA，8514/Aが花盛りですが，バス経由ではこれでもデータ転送がまだ遅い．そこでビデオ専用にCPU直結のローカルバスが注目されています．昨秋のComdexでも何社かマザーボード上にビデオカード専用バスをのせてデモしていました．もっとも，各社規格がバラバラでは困ります．VESAではISA，EISA，MCAなどのシステムの違いに関わらず使えるローカルバスの規格を決めると発表しました．

すでにMylexなどは，EISAバス用のGUIアクセラレータ付きS3チップがのったディスプレイボードを発表しています．しかし，まだ普及していない複雑なEISA，MCAよりも安価に作れるローカルバスボードもこれからの注目でしょう．

★800dpiのレーザプリンタ，そして拡張パラレルポート

さて，西川氏による日本語Windowsドライバのブレイクスルーで，日本でもGUI文化が見えてきました．もっとも，GUI専用のチップがでてこないと使いものにならないMS-Windows GDIもいい加減古い仕掛だと思いますが……．

ともあれ，グラフィックのボトルネックが解決された今，次の壁はプリンタでしょう．プリンタ絡みの話題をいくつか．

GUIが広まるとグラフィックの印字の遅さが気になってきます．今ではページプリンタもとても手軽な値段になってきましたが，その拡張機器の長い経験を持つLaserMasterは，もっとも広く使われているLaserJet III用の800dpiコントローラアップグレードキットWinJet 800*1を発表しました．本来，300dpiのCanonのプリンタエンジンを使っていますが，コントローラを置き換えることにより800dpiの解像度を実現しています．高速にデータを転送し，印字するためにパソコン本体とは専用ビデオインターフェースでつながれます．PostScriptとTrueType，PCLがサポートされている上

ワンラインNEWS
沖電気工業は仏伊合弁の半導体メーカーであるSGS-トムソン・マイクロエレクトロニクスに4MビットDRAMの組み立て生産を委託した．

にWindows専用のドライバが付いてきます．これらはCPU上でラスタライズして，結果のイメージをプリンタのエンジンに直接送り込みます．とくにWindowsドライバでは，本当のWYSIWYGが実現されるわけです．原理的には日本語でも各種出てきた日本語アウトラインフォントと組み合わせれば扱えるはずですね．

それにしても，今のパラレルポートは転送速度が遅すぎます．Laptopでは歴史の長いZenith Data Systemsは，パラレルポートLANアダプタのXircomと組んでEnhanced Parallel Port (EPP) の規格を作り，同社のMastersPort 386SLe Notebookに組み込みました．インテルはすでにEPPを386SLのサポートチップ，82360SLでサポートしています．これにより単に印字データの転送速度が速くなるだけでなく，XircomのLANアダプタのようなプリンタポートに接続する機器もI/Oバスに接続する機器に負けないだけの性能が出せることになります．

＊1:WinJet 800 upgrade kit $995
(8M byte of RAM required)
LaserMaster Corp
6900 Shady Oak Road, Eden Prairie, MN 55344
phone (612) 944-9457

★ FAXモデムの規格について，あるいはClass 2の混乱

このコラムでも，FAX modemの規格の動向についてお伝えしてきました．その中でもEIA/TIA-592となるClass 2は，FAX modemのハードウェアレベルのコマンドセットの標準として大きく期待されています．同様の規格としてClass 1があり，またチップセットメーカーが決めた規格としてSierraのsend only（送信専用）のものが広く使われています．これらは，ヘイズ互換モデムのATコマンドセットを拡張し，どのアプリケーションからでも，対応したFAX modemが扱えるようにするものです．Class 2だけを見ても，すでにRockwellやSierraなど4社以上からチップセットが発売されており，これらを使ったりあるいは独自にファームウェアを作成したりしたFAX modemが広く安価に手にはいるようになっています．

もっとも，このClass 2に関しては現在大きな混乱があります．というのも，これはまだ確定した規格ではなく，現在まだ検討中のものなのです．現在でているClass 2は一昨年のドラフトのSP-2388というものをベースに，見切り発車で作られたものです．ところが現在委員会では，これにかなり変更が加えられたSP-2388-A[*2]が検討中であり，これに少しの変更を加えた形で最終規格となりそうです．そのため現在のClass 2対応ハードウェア，ソフトウェアは大きく変更を求められるか，あるいは事実上の標準としてほそぼそと生き残ることになるでしょう．残念なことです．

なお，現在のSP-2388-A暫定規格の資料は以下から購入可能です．

＊2:SP-2388-A $85
EIA Standards Sales
Phone: 202-457-4966

★ SimCityがシェアウェアに！ 何と25ドル

2年前一世を風靡したゲーム，MAXIS SoftwareのSimCityです．SimEarth，SimAntと続けて発表され，IBM環境ではSimEarthのWindows版まででています．今度オリジナルのDOS版SimCityがなんとシェアウェアになりました．version 1.0がそのままドキュメントも含めて圧縮され，BBSなどを通じて配布されます(600KB以上あります)．これをダウンロードしてユーザーは1週間試すことができます．もし，気に入ってそれ以上使い続けるならばソフトを「購入」しユーザー登録します．お金を払わないことに決めたのなら，コピーをディスクから削除し使用をやめます．

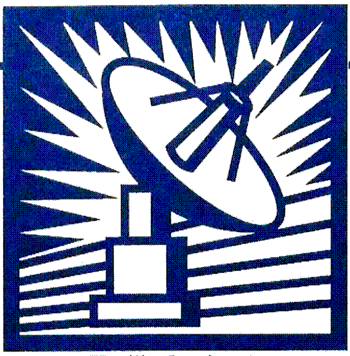
その値段がなんとたったの25ドルです．さらにこれに加えて送料2ドルを加えた27ドルを送金すると，ユーザーはversion 1.1の入ったディスケットと印刷されたマニュアルを受け取れます．

シェアウェアは，個人小切手やクレジットカードなどの送金方法が発達し，それよりも何もパソコンユーザーの意識が高いアメリカでは，すっかり一般的になった販売形態の1つです．それにしてもこれだけ成功した市販ソフトウェアが，これだけの低価格でシェアウェアになるというのは画期的です．

これを企画したのはRWS and Associatesというところです．ShareGame[*3]という名前でより抜きの市販ソフトを，中間マージンをのぞいた安い価格でBBSを通じて販売する試みです．このSimCityのほかにも，いくつか計画中とのことです．ぜひうまくいってほしい企画です．そのためにもこれが気に入ったらぜひちゃんと代金を払いたいものです．そうすれば，これからもよりよいソフトが安い値段で買えるのですから！

＊3:ShareGame Professional Software
RWS and Associates
132 Alpine Terrace
San Francisco, CA 94117

Illustration : NOBUKI FUKUHARA



西川和久の

希望的観測

Multimediaがやってきた!?
Windows MMEの実態を探る

1 盛りだくさんのマルチメディアキット
Creative Labs社のMultimedia Upgrade Kit

例のごとく用もなく秋葉原をうろついていると，T-Zoneでおもしろいものを発見した．「Multimedia Upgrade Kit」である．中身は，

［ハードウェア］
- Sound Blaster Pro
- CD-ROM Drive
- 各種接続ケーブル

［ソフトウェアCD-ROM］
- MME版Windows 3.0
- Microsoft Bookshelf for Windows
- The SelectWave
- JONES in the Fast Line
- SOUNDS

これだけのものがセットになって1パッケージになっている．値段も，入っているものを考えると妥当なところであったし，ちょうどSound Blasterや本体組込み型のCD-ROMドライブも欲しかったので思わず衝動買いしてしまった．もちろん支払いはカードである．(^^;

さっそく会社へ持って帰りインストールを始めた……．PC互換機の本体カバーを開けドライブベイを外しCD-ROMドライブのセット，Sound Blaster ProをISAバスへ差し込み基盤後部のSCSI（モドキ）I/Oから付属のケーブルを使いCD-ROMドライブへ接続して電源コードをつなげば，ハードウェアのセットアップは終了である．残るはソフトウェアのインストールであるが，MMEに対応した米国版WindowsがCD-ROMとして同梱されているので，CONFIG.SYSやAUTOEXEC.BATでCD-ROMドライブがDOSからアクセスできるようにドライバを追加する必要がある．これさえできればあとは簡単で，SETUP.EXEコマンドを実行するだけなのだ．手順はWindowsのインストールとほぼ同じで，いとも簡単に終わってしまった．

2 日本語WindowsへMMEを組み込む
和洋折衷MME版日本語Windowsの完成

普通ならこれでWINと入力すれば英語環境でWindows MMEが使えるのだが，英語が苦手な私にとってそれではつまらない．一応，Windowsが起動することだけを確認し，次なる実験へ移る．

そもそも音や画像データを扱うドライバなど日本語化する必要がないため「そのままDOS/V日本語Windowsでも使えるハズだ!?」と判断し即実行した．SYSTEM.INIやWIN.INIを見比べ，新たに追加されている項目を見つけ出して日本語Windowsへ追加記述，システムディレクトリにロードされているMME関連のファイルをすべてコピーし準備は完了である．かなりいろいろなファイルや項目が追加されているために手こずったが，結果はうまくいった．“和洋折衷MME版日本語Windows”の完成である．日本語版MMEなど待つこ

ワンラインNEWS
ソニーは記憶容量を現行機種の2倍にした5.25インチMOディスクドライブを開発した．これを機に5.25インチMOディスクの標準化を進める．

ともなくマルチメディア対応になってしまった……やはりDOS/Vは偉大なのだ．:-)

3 SB Pro Mixer, Soundrec, Musicbox……MMEの標準機能は??

コントロールパネルを開いてみると，あるはあるは……いろいろマルチメディアに関係しそうなアイコンが追加されている．目新しいところでは「MIDI Mapper」，「Display」，「Drivers」，「Joystick」，「Screen Saver」，「SB Pro Mixer」である．名前から機能は推測できるので詳しい話はしないが，なかでもいちばんMMEらしいのがSB Pro Mixerである．Sound Blaster Proで扱える入出力，たとえば，FM音源／ミュージックCD／サンプラー／マイク入力／ライン入力などの各ボリューム，ハイ／ローカット・フィルタなどがパネルでコントロールできる．Sound Blasterと違いSound Blaster Proは入出力がステレオなのですべてのコントロールが左右に分離し，まるで卓上ミキサーのようである．

MMEではそのデータによって拡張子が定められており，＊.WAVがサンプリング・データ，＊.MIDがMIDIデータである．当然これらのファイルを扱うための標準アプリケーションが追加され，単純に音を鳴らすアプリケーションとしては前者が「Mmcbox」後者が「Mmjbox」である．また「時計」の代わりになるものとして「Alamclk」があるが，先のサンプリング・データを使って指定した時間や1時間ごとに音を鳴らす機能が追加されている．柱時計の“ボーン”という音まであり，なかなかリアルなのだ．WIN.INIのLOAD＝でこのAlamclkを必ずロードしているため，打ち合わせに来た人など，音がなると時計を探してキョロキョロ回りを見る人さえいるありさまだ．%-)

「Soundrec」はサンプリング・データの録音／再生，簡単な味付けなどができるアプリケーションであるが，「ピッチの変更」，「逆再生」，「エコーの追加」などの機能を持ち，波形をオシログラフのように表示するためなかなかおもしろい．最後の極めつけは「Musicbox」である．ミュージックCDをWindows上で操作できるのだ．再生時間表示やプログラミングなどもサポートされ，“いかにもMME!!”と感じることができる．まあ，わざわざWindows上で操作しなくてもDiskManが机の上にあればよいだけの話であるが…….

しかし，残念なことにMicrosoft Bookshelf for Windowsを含め，静止画や動画のデータがほとんどない．さらに，数少ないデータもMPCレベル1に合わせているらしく，すべて16色となっている．せめて1つ位は256色の動画データが欲しかった．音に関しては標準でかなり遊べたため，ちょっと拍子抜けしてしまった．またメモリもけっこう必要らしく，MS-WordとWinTermを中心に細々したアプリケーションを同時に動かしていると，8MBでもメモリ不足で，すぐディスクスワップが始まってしまう．いずれにせよ，CPUパワーも含めかなりシステム資源が必要なのは間違いない．386SX/4MB程度のマシンでは荷が重すぎるであろう．

4 残るはSuper Video Windowsだけ ここまでくれば完璧だ

最近もうひとつ狙っているカードがある．“Super Video Windows”だ!!　これを使うとWindows上にVideo信号で入力した映像がアイコン／窓表示できる．早い話がWindowsで仕事や通信しながらテレビが見られるのだ．縦横比が狂うような窓で表示してもちゃんと画像は追従するため，デモとしてもおもしろい．もちろんDOS/V日本語Windows上で動くのは確認済みである．ここまでくればもう完璧だ．1280×1024，256色／ノーインターレスの日本語Windows上でミュージックCDをコントロール，そしてテレビを窓表示しながらMS-Wordで原稿書き，ちょっとネタにつまったらMIXへ……ウーム，マルチメディアなどこんな不純な動機から普及するのだ!! :-)

Computer Laws

コンピュータの法律

他人にプログラムを作成させた場合の著作権の帰属　その4

◆

熊谷信太郎(弁護士)

■委託した場合の問題点

前回に引き続き，ソフトハウスやメーカーにプログラム作成を委託した場合の法律的な問題点を考えてみたいと思います．

プログラムの発注者がソフトハウスやメーカーにプログラム作成を委託した場合，いったんはソフトハウスやメーカーにプログラム著作権が発生しますが，著作権譲渡契約を締結することにより，著作者人格権以外の通常の著作権は発注者に移転することは前回説明したとおりです．

この著作権譲渡契約に際して注意すべきことは，単に著作権を譲渡するという条項を記載するだけでは不十分だということです．すなわち，著作権を譲渡するという条項だけでは，通常の著作権に属するすべての権利が当然に移転するわけではないのです．通常の著作権のうち，翻訳権，翻案権，および2次的著作物の利用に関する原著作者の権利を譲渡する場合には，契約条項上でそれらを譲渡することを明示しておく必要があり，もし明示していない場合には，それらの権利は譲渡されなかったものと推定するという規定が著作権法に置かれています．

まず，著作権第27条には，翻訳権，翻案権(著作物を編曲し，変形し，脚色し，映画化するなど翻案する権利をいいます)が規定されており，また同28条には2次的著作物(著作物を翻訳し，又は翻案することによって創作した著作物)の利用に関する原著作者の権利が規定されています．

これは，原著作者の著作権を保護するために，原著作物を利用して作られた2次的著作物の利用権限を原著作者にも与えたものです．

原著作者が，翻訳権，翻案権，2次的著作物の利用に関する原著作者の権利も含めて著作権を譲渡しようという場合には，単に，著作権を譲渡するという譲渡契約書を作成するだけでは不十分であり，それらの権利を含めて譲渡するのだということを契約の条項の中に明示しておく必要があります．

すなわち，著作権法第61条第2項は，著作権譲渡契約において，翻訳権，翻案権，2次的著作物の利用に関する原著作者の権利が譲渡の目的として特掲されていないときは，これらの権利は譲渡した者に留保されたものと推定していますので，これらの権利も含めて譲渡したいときは，その旨をはっきり譲渡契約書に明示しておく必要があるのです．

■監修者，校閲者の著作権

次に，プログラムの作成を委託した場合，著作者とならべて著名な人や会社の名前を「監修」「校閲」者としてあげる場合がありますが，この場合には誰が著作権をもつかも問題になります．

著作権法には，監修や校閲に関してとくに規定はありませんので，この場合には，監修者や校閲者がプログラムの作成にどの程度の影響を与え，どのような行為をしたかなど，プログラム作成についての関与の程度によって，結論がかわってきます．

プログラムに権威をもたせるなどの理由から，監修，校閲などの名目で作成者以外の名前を掲げたにすぎず，監修，校閲者がプログラムの内容に立ち入って手を加えていない場合には，監修，校閲者は著作権をもつことはありません．

Illstration : Nobuki Fukuhara

ワンラインNEWS
シャープはLCDの新しい用途別開発例としてペン入力タイプのLCDなど2つの試作品を発表した．今年半ばに商品化する予定．

これに対し，監修，校閲者が自らプログラムの内容を検討し，改訂し，組み直すなど，実質的に著作に相当する行為があった場合には，監修，校閲者も共同著作者となり，プログラム作成者とともにそのプログラムの著作権を有することになります．

もっとも，プログラムの内容を検討したいといっても，内容を調べ，気の付いた点を指摘するといった程度では，実質的に著作に相当する行為があったとはいえないでしょう．

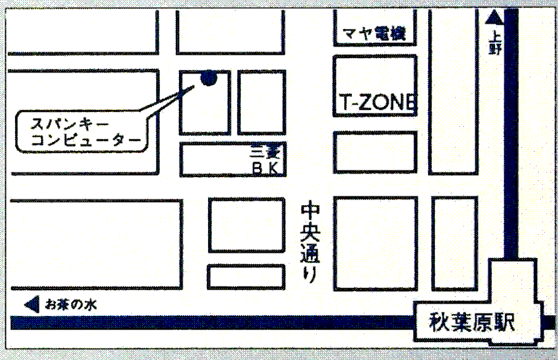
これらの場合，作成者と監修，校閲者との特約で，著作権法の規定と異なる定めをすることはできますから，プログラムの監修，校閲を依頼する場合には，プログラムについて，監修，校閲者がどのような立場に立つかを契約書の上で明確にしておくことが必要です．

アキバ短信

営業時間　10:00〜18:30
定休日　木曜日
TEL　03-5256-7656

スパンキーコンピューター

サポートと品質重視で台湾製互換機を販売



DOS/Vの登場以来，IBM PCとその互換機への注目は日に日に高まりつつある．最近ではパソコン誌に限らず，一般の週刊誌，月刊誌でもWindowsやGUI，DOS/V，互換機といった単語を散見するようになった．もちろんここ秋葉原でも，IBM製品，外国製の安い互換機を販売する店は着実に増えている．たしかに互換機は安い．486/33MHz搭載マシンに，8MBのメモリ，100MBのHDDにディスプレイをつけても50万円そこそことだと聞くと，やはり気にはなる．だが，互換機に関する情報はまだまだ少ないし，買った後のサポートも気になる．やはりまとまったお金を使うなら無難な国産マシンの方がいいのだろうか……．パソコンユーザーのそんな悩みにこたえて，互換機に国産マシンなみの信頼を付けて売りますという店が現れた．それが，今回紹介するスパンキーコンピューターである．

スパンキーコンピューターはIBM PCの互換機メーカーである台湾ユーロテック社の日本総代理店，金亀電子の直営店で，昨年11月にオープンした．同社専務の榊原雄二氏によれば，当初はユーロテック製品のアンテナショップという位置づけのつもりでいたところ，来店する人が予想をはるかに上回ったため，2坪しかなかった店舗を急きょ拡張したのだそうだ．現在取り扱っているのは，ユーロテック社製の互換機本体とディスプレイのほか，IBMの純正パソコン，東芝J3100など各社のパソコン，それに直輸入のIBM用ソフトウェアなどである．

国産マシンを買うのと同じような感覚で互換機を購入してもらうのがこの店の方針．店長はじめスタッフはPCや互換機に詳しい人ばかり，購入時だけでなく，システムアップやアフターサポートまで，さまざまな相談に応じてくれるそうだ．ユーザーそれぞれに最適のシステムをと，場合によってはほかの店を紹介することもあるという．現在もっとも人気のある製品は？　という質問には，486/33MHz搭載のユーロテック社製互換機という答が返ってきた．また，2月末からはすべてのユーロテック製品に，高解像度ドライバ（1024×768ドット）をバンドルする予定であるという．現在は本体中心の品ぞろえだが，今後は周辺装置やボード類も取り扱っていきたいとのことだ．

なお，スパンキーコンピューターでは，内側のボードが見える，匡体の透明な486マシンを展示中である．不幸にして，まだ1度も486CPUの実物を見たことのないという方は，1度のぞいてみてはいかがだろう．

●プログラマの仕事場 12

PWS Programmer's Working Space

アルパソフト株式会社

夢のあるWindowsアプリケーションの開発をめざして

1991年は，まさにWindows 3.0時代の幕開けであった．一般のパソコンユーザーまでが競ってWindowsを買い求めたが，実際にはWindowsに対応した日本語アプリケーションはまだごくわずかだ．各ソフトハウスにも対応が迫られる中，アルパソフトは，すでにWindows3.0対応のアプリケーション「Win-Tool」を開発・発売した実績をもつ．台東区浅草橋にある同社を訪ねて，代表取締役の杉山高男氏にお話をうかがった．

Windowsアプリケーションを1から自社開発

アルパソフトは，1990年5月に設立されたばかりのまだ若い会社だ．社員は9名．全員がソフトウェア技術者である．同社の業務の内容は，受託によるソフトウェア開発と自社のオリジナルのパッケージソフトウェアの開発・販売である．受託のソフトウェアは，システム系のものやISDNによる通信関連のものなどが中心だ．パッケージソフトの第1弾として，91年7月には「Win-Tool Ver.1.0」を市場に送りだした．同社のパッケージソフトウェアに対する考え方を，杉山氏は次のように語っている．

「私どもでは，アプリケーションの自社開発はWindowsがメインです．Windows3.0用——次は3.1，そのうちWindowsNTが出てくれば，それに合わせて順次開発していきたいと考えています．また，人数も少ないですし，若い社員が多いので，実際に自分が使いたいと思うようなソフト，作りたいソフトを出していきたいと思います．実際の作業というのは，現実と理想を分けて考えています．理想というのは『こういうソフトを作りたい』という夢です．でも，実際には利益をあげなければなりませんから，受託のほうも一生懸命やるわけです．いまはまだ，受託7割，オリジナル3割ですが，私としてはオリジナルの自社開発のほうを80パーセントくらいまでもっていきたいですね．」

「Windows対応のアプリケーションを自社で1から開発しているところは，日本でもまだ9～10社ぐらいしかありません．だいたい開発元は外国で，それを移植しているという会社が多いですから．Windowsアプリケーションをはじめから自社で開発できるということが，まず私どもの特徴ですね．もうひとつには，Windowsがらみの周辺デバイス関係もやっています．DDKというWindowsのデバイスドライバ用の開発キットを使って，ISDNのパソコンボードをWindowsから使えるようにデバイスドライバを開発したいと考えています．」

WindowsSDKにも書いてないテクニックがあるはず

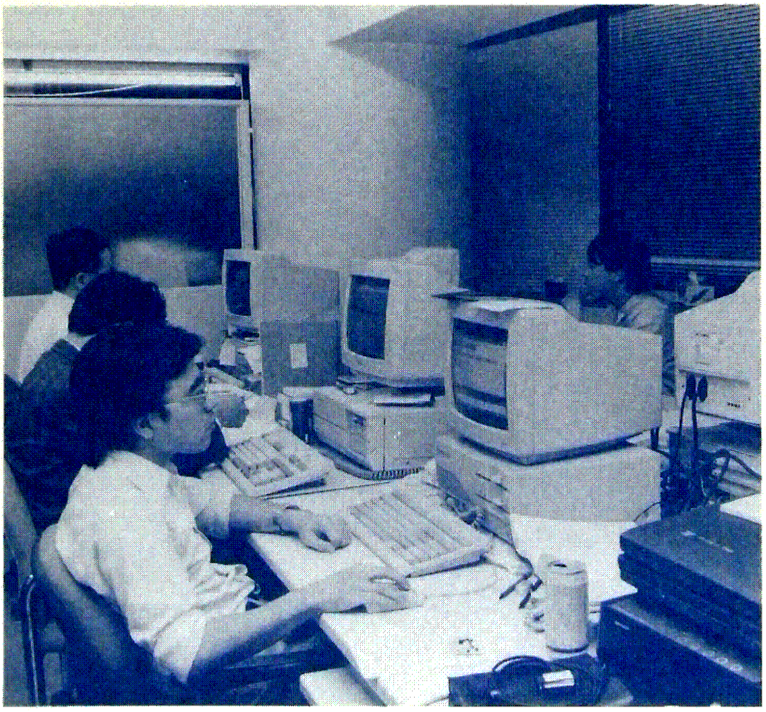
ワンラインNEWS
松下電器産業は自社製のスーパーコンピュータ「アデナート」を東京，名古屋両大学にも貸与し，アデナートを利用する機関は4か所となった．

日本でもマイクロソフト社がWindowsアプリケーション開発者（ISV向け）のためのサポートを行っているが，細かい情報をつかむにはまだ不足だと杉山氏はいう．

「Windowsのアプリケーションを作ろうと思ったら，たいていのエンジニアはまずSDKを買ってきて，プログラマーズガイドを見ます．それで作るのは最低限の形式です．本当にWindowsアプリケーションを開発するには，テクニックというのがあるはずなのです．マイクロソフトから公開されないファンクションというのもいろいろありますし，日本にはない技術情報も，米国では出回っていたりするのです．日本の現状では，もう少しおもしろいテクニックを使ったソフトを作ろうとすると，よくわからないということになりがちです．SDKにある情報をたよりに，形式だけで作ってしまうと，米国のソフトと比較したときに『遅い』『何か違う』ということになってしまいます．日本ではそういった情報がなかなか入ってきませんね．」

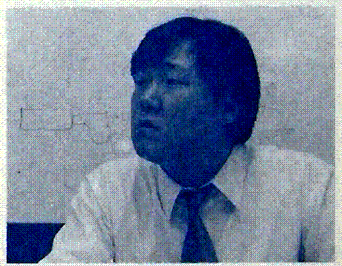
パソコンLANで結ばれた開発環境

同社の開発環境は，ひとりにつき1台ないしは2台のパソコンがMS-NETWORKSで結ばれているというもの．サーバマシンには，600MBのMO，40MBのハードディスクとレーザープリンタが接続されている．MOには各人用のワークエリアのほか，各種の開発ツールや言語処理系，開発中のアプリケーションプログラム，納品ずみの受託ソフトなどがおさめられている．ローカルのマシンはNECのPC-9800シリーズまたはAXパソコンで，それぞれ4～8MBのメモリを搭載し，100MBのハードディスクを接続している．杉山社長によれば，将来的には開発マシンを比較的価格の安くなっ



About Company

アルパソフト株式会社．1990年5月設立．代表取締役杉山高男．SI，LAN，ISDN関係を含めたソフトウェア，システムデバイスの受託による設計・開発，アプリケーションソフトウェアの開発，コンピュータ導入のコンサルティングをおもな業務とする．電話03-3863-5977㈹



代表取締役　杉山高男氏

Location

JR総武線で秋葉原の隣，浅草橋駅東口を出て，浅草方向へ5分ほど歩いたところにある．この辺りは造花や飾りものを売る店が多く，アルパソフトのあるビルも1階は造花などを売る店が入っている．

Products

「Alpasoft Win-Tool Ver.1.0」（FMR，PC98，PS/55，AX，J-3100シリーズ対応，9,800円）

MS-Windows Ver.3.0のファイルマネージャ的な機能をもつユーティリティソフト．ネットワークドライブやRAMディスクにも対応，複数のツリーウィンドウの同時表示，複数ファイルの内容表示，ダンプが可能．また，アクセサリボタンにアプリケーションプログラムを登録することによって，簡単にアプリケーションが起動できる．

「Estella」(FMR, PC98, PS/55, AX, J-3100シリーズ対応, 52,800円)
カード型データベース，文書編集システムなどをもつWindows 3.0対応の統合型アプリケーション．データベース機能は，T型，カード型，リスト型を組み合わせ，データ編集，複数項目に対して条件を指定した検索，辞書順ソートが行える．文書編集システムは，印刷イメージを画面上で確認できるWYSIWYGを実現し，より高度な編集が行える．4月発売予定．

てきたPC/ATコンパチ機にしたいという意向もあるという．

プログラミング言語は，MS-C6.0，MASM5.1，Borland C++，Lattice Cを使用している．Windowsアプリケーションの開発にはBorland C++を使用しているとのことだった．

技術力アップをめざして週1度の講習会

Windowsアプリケーションを作れるプログラマはまだそれほど多くはない．人材を確保するのは，たいへんではないだろうかと杉山氏にたずねてみた．

「現在アルバイトを含めて10人の技術者がいますが，そのうちWindowsのアプリケーションが組めるのはまだ3人です．実際のところ求人を出してもなかなか人が集まりませんね．汎用機ではプロだけれども，パソコン――MS-DOSやWindowsの開発を一度もやったことがないといった人は，すぐに仕事をしてもらおうと思っても無理ですからね．ちょうどこちらで求めている人を探すのは，なかなか難しいです．ですから，お互いにもっている情報を活用するために，週に1度は社内で講習会を開いています．Windowsのプログラミング，アセンブラ，Cなどのテーマについて，夕方の5時半から1時間程度で，各人が自分の慣れていることを発表するわけです．それも，ただ話を聞くというだけではやはり身につきません．簡単なものでも，自分でプログラムを作って動かしてみなければ絶対覚えられません．プログラムの技術では，やはり自分で作ってみることが大切です．」

今後のWindowsはマルチメディアがキーになる

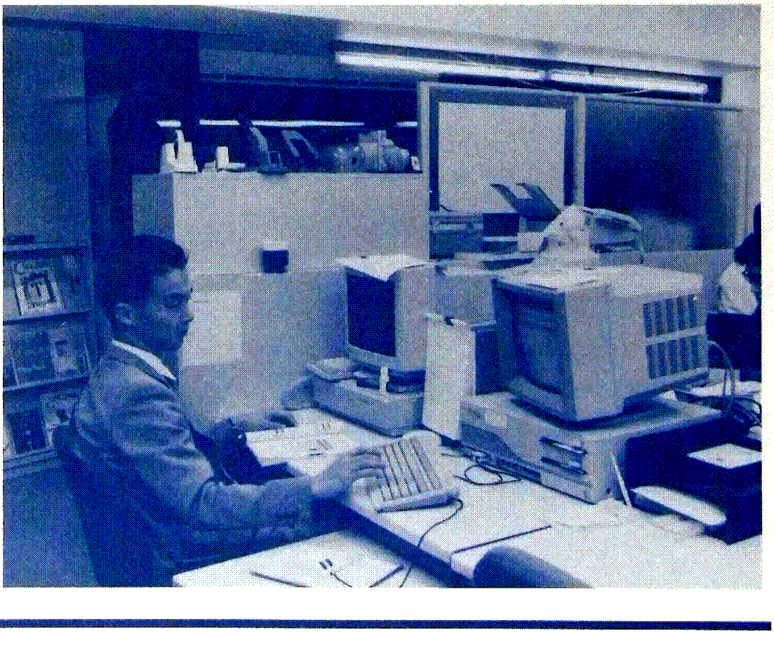
今後アルパソフトがどのようなソフトウェアを開発していこうとしているのか，杉山氏に将来の展望を聞いてみた．

「Windowsは，現在3.0，それが今後3.1，WindowsNTへと進んでいこうとしているわけですが，それにしたがってマルチメディアへの対応が進んでいくでしょう．私どもでは，基本的に情報関係に絞ったアプリケーションやマルチメディア対応のアプリケーションを考えています．」

「現在，すでにビデオやカメラなども一体化したシステムへどんどん移行している状況です．でも，実際に大きなメーカーさんなどの話を聞いてみると，マルチメディアが内部の各事業部で別々に進められていてうまく連携がとれていないということもあるようです．つまり，会社の内部で，カメラの事業部，ビデオの事業部，パソコンの事業部などに分かれて，それぞれバラバラにマルチメディアをやっている．そのために，なかなか成果を上げられない，話が進んでいかない，といった問題もあるようです．」

なお，今年はWindowsアプリケーション第2弾としてカード型データベース&文書編集システムの発売も予定しているという．いずれは，通信やマルチメディア技術を高度に利用して，ビジネスの現場で活躍できるWindowsアプリケーションを作っていくという目標をもっているようだ．

アルパソフトの今後の活躍ぶりに注目したい．



ワンラインNEWS
スリーコムは日本電気のパソコンPC-9801シリーズ用のLANアダプタボードを発売した．スリーコムは昨年8月に米3Comが設立した日本法人．

INFORMATION PAGE

●イベント

■2月20日(木)～2月22日(土)
MACWORLD Expo/TOKYO
Macintosh関連製品の展示とコンファレンスが行われる。
会場：日本コンベンションセンター(幕張メッセ)
入場料：【前売り】33,000円(コンファレンス)
2,000円(展示会・3日間通し)
【当　日】38,000円(コンファレンス)
1,500円(展示会)
問い合わせ先：日本工業新聞社事業部
MACWORLD Expo/Tokyo事務局　TEL 03-3292-3561

■2月26日(水)～2月28日(金)
LAN Expo '92
LAN(ローカルエリアネットワーク)関連製品の総合展示会。会場内のブースをLANで結び，LANを直接体験することができる。
会場：東京国際見本市会場(東京・晴海)
入場料：1,000円
問い合わせ先：(株)マーコム・インターナショナル
LAN Expo事務局　TEL 03-3403-8515

■2月26日(水)～2月28日(金)
ボード・コンピュータ'92
CPUボード，メモリボードなどの産業用ボード・コンピュータの専門展示会。
会場：サンシャインシティコンベンションセンターTOKYO(東京・池袋)
入場料：1,000円
問い合わせ先：(社)日本能率協会
TEL 03-3434-1377

■3月13日(金)
第8回ソフトウェアコンファレンス
研究開発に伴い開発されたソフトウェアに，発表，討論，評価の場を提供，有効利用としての流通，共同開発の促進などを目的に，特別講演，ソフトウェア発表会などのコンファレンス，セッションなどが行われる。
会場：(財)大阪科学技術センター(大阪市・西区)
参加費：チュートリアルセッション 3,000円(一般)，1,000円(学生)
コンファレンス，発表会などは無料
問い合わせ先：(財)大阪科学技術センター
TEL 06-443-5321

●海外イベント

■3月6日(金)～3月10日(火)
台北テレコム'92
アンテナシステム，データ通信，ネットワーク関連製品などを一堂に展示。同時に，国際通信技術に関するセミナーも開催される。
会場：台北世界貿易センター
問い合わせ先：遠東貿易サービスセンター
TEL 03-3407-9711

■3月10日(火)～3月12日(木)
Multimedia and CD-ROM
CD-ROMハードウェア，ソフトウェア，マルチメディア関連製品の展示とコンファレンスが行われる。
会場：San Jose Conventioncenter(米国・カリフォルニア州)
問い合わせ先：カーナーズエクスポジションジャパン(株)　TEL 03-3349-8501

●セミナー

■3月10日(火)～3月12日(木)
オブジェクト指向設計とC++セミナー
オブジェクト指向技術によるC++プログラミングを体験学習する。
会場：麹町会館(東京・千代田区)
受講料：190,000円(3名以上，2講座受講者に割引あり)
問い合わせ先：ソフトウェアテクノロジージャパン　TEL 03-5689-3536

Microsoft University

■2月24日(月)～2月28日(金)
MS OS/2 Applications Programming
OS/2アプリケーション開発のための上級コース。
受講料：180,000円(5日間)

■2月24日(月)～2月28日(金)
3月9日(月)～3月13日(金)
3月23日(月)～3月27日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース。
受講料：180,000円(5日間)

■2月25日(火)～2月28日(金)
MS OS/2 Device Drivers
OS/2デバイスドライバの機能と構造を解説する。
受講料：150,000円(4日間)

■3月2日(月)～3月6日(金)
Microsoft Windows Applications Programming
Windowsアプリケーション開発のための上級コース。
受講料：180,000円(5日間)

■3月2日(月)～3月6日(金)
MS OS/2 Presentation Manager Programming Environment
OS/2 PMアプリケーション開発のための入門コース。
受講料：180,000円(5日間)

■3月9日(月)～3月13日(金)
Microsoft LAN Manager Programming Environment
LANマネージャアプリケーション開発のための入門コース。
受講料：150,000円(5日間)

■3月12日(木)～3月13日(金)
C言語入門
QuickCを使ったC言語の入門コース。
受講料：60,000円(2日間)

■3月17日(火)～3月19日(木)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース。
受講料：90,000円(3日間)

Microsoft Universityは，OS/2，Windowsアプリケーション開発者のために定期的に開催されているセミナーです。
問い合わせ先：富士ソフトウエア(株)
品川教育センター　TEL 03-5461-8296

NEWS & NEW PRODUCTS

オオツカ商事，レーザプリンタ用プログラム開発支援ツールを発売

㈲オオツカ商事（ユーザーサポートセンター：TEL 0878-51-1807）は，レーザプリンタに必須のプログラム開発支援ツール＆Cソース「NAPRICO（ナプリコ）リコー編」を発売した．レーザプリンタを完全ネイティブモードで動作させ，ネイティブ制御に必要な基本関数のほか，NECのPC-PR201H2用制御コードを完全にネイティブモードへ変換するパーツ関数を含み，エミュレーションカードでは不可能なネイティブモード独自の高品質な印字が可能となる．レーザプリンタのパネル設定をソフト的にパソコンからコントロールするパネル遠隔操作構築パーツ，市販ソフトにも有効な常駐機能，装飾書体・超拡大文字のサポート，アウトラインフォント自動設定機能などをもつ．また，C言語ソースを標準添付している．対応プリンタはリコーのLP3320-SPmk II，LP-2310-SP5．適応機種はNEC PC-9801，98NOTEシリーズ（XL・LT・HA・ハイレゾモードを除く），EPSON PC-286/386シリーズ．価格は52,800円．

西友ソフトウェアー，PS/55，DOS/Vマシン用C言語開発支援ツールを発売

西友ソフトウェアー㈱（TEL 03-3257-0670）は，IBM PS/55，DOS/Vマシン用「C言語開発支援ツール」を発売した．このライブラリには，キー入力，画面制御，メニュー，ウインドウ，グラフィック，マウス，プリンタ，PC-DOS，RS-232Cの9つのツール関数が含まれている．価格は420,000円で，ソースが付属している．また同時に，PC-9800シリーズ対応のC言語CAIソフト「C教授の基礎講座」をバージョンアップした．初心者でもゲーム感覚で楽しくC言語を学ぶことができ，各章の練習問題で，自分の理解度を確認しながら学習することができる．今回のバージョンでは，新たに，関数についての章などが追加された．価格は14,800円．

マイクロフォーカス，Micro Focus日本語 COBOL/2をOEM供給

マイクロフォーカスジャパン㈱（TEL 03-3486-7791）は，日本ユニシス㈱へ同社のU6000/65シリーズ対応の「Micro Focus日本語COBOL/2」をOEM供給する．「Micro Focus日本語COBOL/2」は，MS-DOS，MS OS/2，UNIXに対応し，単独またはネットワーク化されたパーソナルコンピュータ，ワークステーション，メインフレームにおよぶ広範囲な分野で，多岐にわたるCOBOL標準，およびダイアレクト（IBM OS/VS，IBM VS COBOL IIなど）をサポートする環境を提供する．

ボーランド，「dBASEIV LAN Pack 5ユーザー版」を発売

ボーランド㈱（TEL 03-5489-4867）は，リレーショナルデータベースソフト「dBASEIV」にマルチユーザー環境を提供する「dBASEIV LAN Pack 5ユーザー版」を発売した．8段階のユーザーアクセスレベル，パスワードプロテクション，データの暗号化などのセキュリティ機能，データベースファイルの変更に対してトランザクションログファイルを作成し，更新中に割り込まれたデータベースファイルを復元することができるトランザクション処理機能，複数ユーザーからのアクセスに対してデータを護る自動レコード・ロック機能，データベース情報を自動的に更新し，更新された際にはその変更を自動的に表示するリフレッシュ機能などをもつ．価格は298,000円．ほかに，10ユーザー版（477,000円），40ユーザー版（1,550,000円）もある．また同時に，IBM PS/55対応版「dBASEIII Plus 2.0J」の価格を，現行の268,000円から198,000円に引き下げた．

ワンラインNEWS
アイ・オー・データ機器は「GA-1024i」のオプションキットとして，標準解像度のディスプレイで高解像度表示を可能にするアダプタ「GA-OPI」を発売．

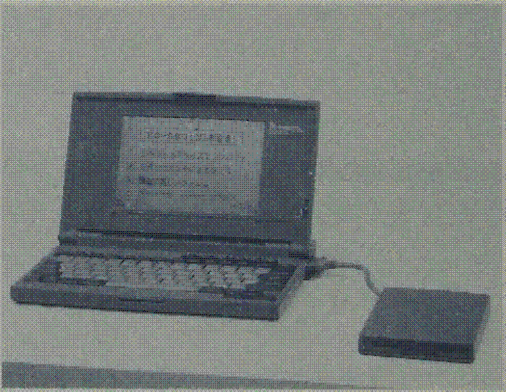
NEWS & NEW PRODUCTS

ナンタケット，AX対応Clipper 4.5日本語版を発売

ナンタケット㈱（TEL 03-3725-8007）は，PC-9800シリーズ，FMRシリーズに続き，AX対応版の「Clipper 4.5日本語版」を発売した．「Clipper」は，アプリケーションの標準開発言語として，とくにネットワーク上での，強力で高度な標準的なシステム構築，その実行速度，開発効率，プログラムソースコードのセキュリティ，他言語とのインターフェースの点などで高い評価を得ている．ネットワーク上で使用する場合にも，LANPACKやランタイムなどの余分な費用はいっさい必要ない．また，同時にClipper 4.5日本語版用のアプリケーション開発用ジェネレータ「ClipGen」（PC-9800シリーズ用，58,000円）も発売した．

日本電気，16・32ビット98NOTEの新機種などを発売

日本電気㈱，日本電気ホームエレクトロニクス㈱（NECパソコンインフォメーションセンター：TEL 03-3452-8000）は，16ビットノートパソコン「PC-9801NL（98NOETライト）」（写真）と32ビットノートパソコン「PC-9801NS/T（98NOTE SX/T）」を発売した．携帯性を重視した「98NOTEライト」は，外付け型の3.5インチフロッピーディスクドライブ，内蔵RAMドライブを標準添付，本体重量1.3Kg，最大バッテリ駆動時間8.4時間，ジャストA4サイズを実現している．価格は238,000円．「98NOTE SX/T」は，「PC-9801NS/E」の後継機で，現行機にくらべ，処理速度1.25倍，バッテリ駆動時間はフロッピーモデルで3.8時間，約12%増加している．フロッピーディスクモデル（288,000円），40MB HDD内蔵タイプ（398,000円），80MB HDD内蔵タイプ（458,000円）の3モデルがある．また同時に，80486SX（16MHz）を内蔵し，現行機にくらべ，処理速度1.6倍，本体前面にファイル装置を増設できる「ファイルスロット」を装備した32ビットデスクトップパソコン「PC-9801FA」（458,000円～648,000円）も発表した．



日本電気，業界最高速ラップトップ型ワークステーションを発売

日本電気㈱（ワークステーション事業部：TEL 03-3798-6288）は，RISC型マイクロプロセッサVR3600（25MHz）を搭載し，ラップトップ型としては業界最高速の演算速度28MIPSを実現したワークステーション「NECスーパーステーションEWS4800/120LT」を発売した．解像度1280×1024ドット，13インチTFT白黒液晶ディスプレイを採用し，メモリの最大容量48MB，内蔵磁気ディスク装置の記憶容量418MBと，ラップトップ型としては業界最大．OSは，「UNIX System V Release 4.0」，「UNIX System V 日本語環境共通規約」，「MIPS-ABI」に準拠している．最小システム（本体，メモリ16MB，内蔵磁気ディスク装置418MB，マウス，基本ソフト）の価格は1,750,000円．

メガソフト，自動音声応答 FAXサービスシステムを発売

メガソフト㈱（本社：TEL 06-386-2043，東京事務所：TEL 03-5261-7077）は，パソコンFAX

NEWS & NEW PRODUCTS

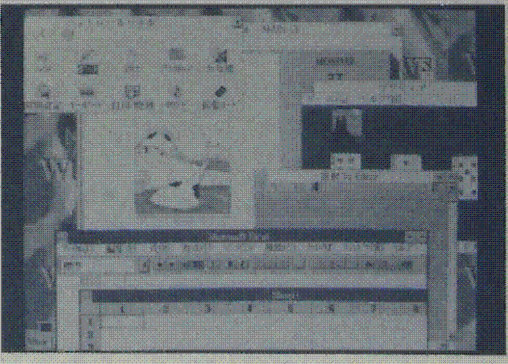
「STARFAX」をベースにした自動音声応答FAXサービスシステム「STARFAX EXPRESS」を発売した．「STARFAX EXPRESS」に，G3FAXからダイヤルし，音声による案内に応じて該当番号をダイヤルプッシュするだけで，製品カタログなどの必要な情報をFAXで受信することができる．自動運転による24時間サービスで，夜間，休日などの業務時間外でも，365日，24時間フル稼働の情報サービス体制を構築することができる．対応機種は，PC-9800シリーズ（デスクトップタイプ386マシンが望ましい）で，FAXボード，32ドットフォントボード，録音・再生用電話機などが付属している．価格は1,800,000円．

エプソン，アウトラインフォントアクセラレータボードを発売

セイコーエプソン㈱，エプソン販売㈱（エプソンインフォメーションセンター：TEL 03-3377-3531）は，プリンタに出力されるものとまったく同じものをディスプレイ上に表示し，WYSIWYGを可能にするアウトラインフォントアクセラレータボード「Fontアクセラレータ OFP-3000」を発売した．日本語MS-Windows 3.0を使用し，エプソンPCシリーズ，NEC PC-9800シリーズのパソコン上で，ボード上に標準搭載した明朝体とゴシック体，およびオプションで供給される各種書体のアウトラインフォントを拡大・縮小，修飾し，同時にディスプレイに表示させ，そのままのイメージでプリンタに出力することができる．パソコン上で展開したアウトラインフォントをそのままプリンタに転送するため，アウトラインフォントを搭載していないプリンタやシリアルプリンタでも，美しい文字を印刷することができる．また，MS-DOSで作成されたテキストファイルを編集，印刷することのできる「MS-DOSテキストファイル簡易レイアウタ」も用意されている．価格は，69,800円．

CFC，海外のビデオボードを日本語Windowsに対応させる製品を開発

シーエフコンピューティング（CFC）社は，米国などで販売されているビデオアダプタボードに付属しているWindows用ディスプレイドライバを，DOS/Vの日本語Windowsで使用可能にするソフトウェアを開発した．このソフトウェアは，Windows Display Dispatch Driver（DDD）で，日本語Windowsにもオリジナルの Windowsと同等の利用環境を実現する画期的な製品である．海外では，Windowsの描画スピードを高速化するグラフィックアクセラレータチップを搭載したり，VGAを越える精細画面（800×600，1280×1024など）や多色表示（256色，フルカラーなど）を実現するボードが数多く発売されている．従来これらのボードは，日本語Windowsでは利用できなかったが，DDDにより障壁がクリアされたことになる．DDDは，英語版ディスプレイドライバをディスパッチし，日本語WindowsのWIFEマネージャに対応させる．これによって，ディスプレイドライバを書き直すことなく，そのまま日本語Windowsで利用することが可能となる．DDDは，T-ZONEなど800×600日本語ディスプレイドライバを販売していたパソコンショップを通して4,000円程度で販売される．CFCでは，DDDのエンジンを組み込んだ完全な日本語ドライバの開発も行う予定．



1024×768　256色 で 日 本 語 Windows が 動 く（ET4000を搭載したグラフィックボードを使用）

ワンラインNEWS
ソフト開発会社のダットジャパンは同社の発売している「1-2-3カード」のハングル語版を韓国で発売した．同ソフトは「Lotus 1-2-3」の機能拡張を行う．

NEWS & NEW PRODUCTS

イースト，Excel用ADD-INカード型データベースソフトを発売

イースト㈱（TEL 03-3374-1980）は，表計算ソフトMicrosoft Excelのアドインソフトで，Excelのデータをカード形式で表示，印刷，入力，編集が行えるカード型データベースソフト「CARD for Excel」を発売した．カード形式での入力画面のレイアウトや伝票，フォーム紙などに出力するための出力帳票デザインを画面上でWYSIWYGで行うことができる．1画面がExcelの1行に対応，カード形式でのデータ入力が行え，Excelの表示，編集機能に追加して，かな漢字変換のON/OFF，入力範囲のチェックなどの機能を使うことができる．項目間のコピー，ペースト，データ一覧のメニュー表示，TableLookUpなどもExcelでの定義と連動して使うことができ，カード型での検索も可能．伝票，ラベル印刷，ハガキ，封筒などの印刷も簡単なメニュー選択で行うことができる．日本語Windows 3.0，Microsoft Excel 3.1に対応．価格は24,000円で，春頃出荷の予定．

東芝，米国サン・マイクロシステムズのOS「ソラリス」採用を決定

㈱東芝（情報処理・制御システム企画室：TEL 03-3457-8145）は，米国サン・マイクロシステムズ社が開発したOS「ソラリス2.0」を，同社のパソコン「3100シリーズ」およびワークステーション「ASシリーズ」の共通OSとして採用することを決定した．「ソラリス」はUNIXをベースにウィンドウズ機能，ネットワーク機能を組み込んだOSで，RISCチップSPARC，インテル社の80386以上の高位マイクロプロセッサに対応している．新OSの採用により，IBM PCアーキテクチャに基づくパソコンの上位機種とSPARC LTなどのワークステーションのより緊密な融合をはかる．提供時期は，ワークステーションが平成4年上期，パソコンが平成4年下期の予定．

日本IBM，パソコン用周辺機器の新流通施策を採用

日本アイ・ビー・エム㈱（TEL 03-3586-1111）は，PS/55Zシリーズ用周辺機器の品揃えを拡充するため，他社製品の取り扱いを行う新たな制度として，「VLH（ベンダー・ロゴ・ハードウェア）」を採用することを発表した．パソコンの利用環境を整えるため，パソコン単体だけでなく，ソフトウェアや周辺機器の品揃えを拡充し，ユーザーにあったソリューションを提供していくことを目的としたもので，最初のVLH製品として，ゲーム用のサウンド・ボード「SOUND BLASTER V2.0」（34,800円）とアダプタボード，ケーブル，デバイスドライバをセットにした「SCSIアダプターセット（ATバス）」（41,800円）を扱う．

アップル，ソニー「AppleTalk」のライセンス契約で合意

米国アップルコンピュータ社とソニー㈱（NEWSインフォメーションセンター：TEL 03-3448-3553）は，アップル社のネットワークプロトコル「AppleTalk」のライセンス供与に関して合意し，契約を締結したことを発表した．AppleTalkのライセンス供与を受けるのは，国内コンピュータメーカーとしてはソニーが初．この契約に基づき，ソニーはAppleTalkを同社のワークステーション「NEWS」に移植し，1992年夏をめどに日本国内で商品化を行う予定．AppleTalkは，アップル社が開発したMacintosh用プロトコルで，ユーザーが容易にネットワークを構築し，プリンタなどの周辺機器や各種ファイルを共有することができる．これにより，NEWSとMacintoshとの接続が容易になり，情報や周辺機器の共有が可能になる．

ワンラインNEWS
日本電気は同社の4MビットDRAMの生産量が，92年末に月間400万～500万個規模に達するとの見通しを明らかにした．4M需要の立ち後れから1年ほどずれこんだ．

Programmer's Program

Cソースファイルのポータビリティをチェックする

PC-lint

lintはCで書かれたプログラムソースファイルのポータビリティをチェックするツールで，UNIXのプログラマであれば知らない人はないだろう．プログラム自体やプログラムの標準ライブラリに対する部分が正しいかどうかを確かめ，ポータビリティに対するエラーチェックをし，プログラムがガイドラインに沿ったものであるかどうかを調べてくれる．

lintを使うことで，美しくポータビリティに優れたプログラムを書くことが可能になる．PC-lintは，このUNIXツールのMS-DOS版である．PC-lintはANSI Cに対応し，また各コンパイラ独自の拡張もサポートしている．

インストール

パッケージには5インチのフロッピーディスク1枚とバインダに閉じられた英文のマニュアルが入っている．PC-lintはIBM PC/ATおよびその互換機で走らせることを前提としている．つまりフロッピーディスクは2DであるのでPC-9801シリーズで使用する場合は注意が必要である．まず適当なディレクトリを作成し，フロッピーディスクに含まれるすべてのファイルをコピーする．

このディスクの中には，マニュアルの付録Bに収録されるプリントユーティリティpr.exe，個々のコンパイラのオプションファイル（co-*.lnt）やコンパイラの差異を吸収するための標準ライブラリ定義ファイル(sl-*.c)，そして本体であるlint.exeがある．read.meにはマニュアルには載っていないアップデート情報が含まれているので，マニュアルを読んだ後，必ず読むようにしたい．

次にインストールプログラムを走らせる．MS-DOSのコマンドラインから

```
C> INSTALL
```

と入力し，あとは画面の指定にしたがって設定するだけである．このプログラムは使用しているコンパイラに合ったプロジェクトファイルを作ってくれる．

サンプルファイル

残ったファイルはPC-lintのサンプルファイルである．カレントディレクトリをPC-lintの入ったディスクまたはディレクトリに移動し，MS-DOSのコマンドラインから

```
A> LINT FILEA FIEEB
```

と入力すれば，画面にリスト1のように表示されるはずである．この例では最上段はPC-lintのバージョンを表し，3，5行目は処理されるモジュール名である．実際にPC-lintが複数のファイルを処理しているときは，ほかには何も表示されない．これらのメッセージはオプションによって消したり，より詳しくすることができる．6行目からがメッセージである．

通常メッセージは，一連の行から構成される．リスト1では，7行目がエラーの発見された実際のソースで，6行目の「_」によってエラーを発見した場所が示されている．8行目は処理しているファイル名と行番号（この場合は5行目に問題があったことを示している），それに続いてメッセージの内容が示され，9行目には，発見した問題点に関係するファイルとエラーメッセージが示されている．

コマンドラインの指定

MS-DOSのコマンドラインから

```
A> LINT
```

と入力すればヘルプが表示される．

環境変数LINTにオプションを設定しておくことができる．たとえばMS-DOSのコマンドラインから

```
A> SET LINT=-ml
```

と入力しておけば，PC-lintを実行す

ワンラインNEWS
SGS-トムソン・マイクロエレクトロニクス傘下にある英インモスはRISC型32ビットMPU「IMST805-G30S」（クロック周波数30MHz）の量産を開始．

るたびに『-ml』が設定されることになる。

また「.LNT」という拡張子をもったファイルにコマンドラインオプションを記録して呼び出すことができる。たとえば

```
-e7?? co-lc3 sl-lc3
m1 m2 m3 m4 m5
```

というテキストファイルを作成して，PROJECT.LNTというファイル名で保存する。MS-DOSのコマンドラインから

```
A> LINT PROJECT
```

と入力すれば，コマンドラインから

```
A>LINT -e7?? co-lc3 sl-lc3 m1 m2
m3 m4 m5
```

と入力したのと同じことになる。

C言語で書かれたプログラムは移植性が高いといわれているが，実際にはコンパイラによる独自の拡張機能やOSの機能の差などによって，移植性は著しく下がってしまっている。ANSI Cに従って記述することである程度の移植性は保証されるものの，コンパイラによってはコンパイルエラーは出ないのにプログラムの動作がおかしいといった問題が生じることも少なくない。このような移植性の問題解決に，PC-lintは強いみかたになってくれる。

PC-lintは文法上の誤り以外に，関数の引数のミスマッチ，ポインタの初期化し忘れ，未使用の変数，キャスト，評価順序，演算順序，ネスト，セミコロン，プリプロセッサの例外的使用，列挙型，printf/scanfの引数，未初期化・未使用の構造体，ANSIキーワードの認識，unsigned/signedの検査など，多岐にわたるエラーチェックを行ってくれる。

複数のコンパイラを使用したり，複数のOS上のプログラムを単独のソースで管理するような場合に，PC-lintはなくてはならないツールになるだろう。

PC-lint V.4.00は，Microsoft C，Borland Turbo C，Lattice C，Power C，Datalight C，Watcom Cなど21種類以上のコンパイラに対応している。

■PC-lint V.4.00

販売：(株)サザンパシフィック(TEL.045-314-9514)／価格：29,800円／適応機種：MS-DOSマシン／動作環境：MS-DOS version 2.0以降／必要メモリ：256KB／供給メディア：5インチ2Dまたは3.5インチ2DD

リスト1

```
1:PC-lint Vers. 4.xx, Copyright Gimpel Software 1986-1990
2:
3:--- Module:   filea.c
4:
5:--- Module:   fileb.c
6:
7:    if( f(3000) ) return;
8:fileb.c  5  Warning 516: f has arg. type conflict
9:    (no. 1 -- long vs. int) with line 2, file filea.c
```

注：行番号は説明のためにつけたもの，普通は出力されない。

オンラインソフト採集帳 15
列車発車時刻案内板 TRAINS

PC-9801用の日本語MS-Windows Version 3.0 (以下Windows)が発売されて，まもなく1年になる．しかし，Windows上で走るアプリケーションは数えるほどしかない．さらにフリーソフトウェア，しかも国産となると皆無に等しいのが現状ではないだろうか．TRAINSはそんな，数少ない国産ソフトのひとつである．

駅には，これからその駅を発車する列車についての案内板があり，今度発車する列車の種別・列車名・発車時刻・行き先・乗り場などが表示されている．「TRAINS」は，この列車発車時刻案内板をWindows上で実現するものである．ドキュメントでは「なんら実用性がありません」と謳っているが，必要路線の時刻表を入力すればリアルタイムな時刻表として役立つのは間違いない．

TRAINSとは

TRAINSはパソコンのシステムクロックの現在時刻をもとにして，これから発車する列車についての種別・列車名・発車時刻・行き先・乗り場を駅の案内板のようにWindows上に表示するプログラムである．この案内板には，最大10本の列車の情報が表示される．最上段が今度発車する列車で，以下それにつづいて発車する順に表示される．なお起動時には，最初の2本の列車のみを案内板に表示するようにウィンドウサイズが設定されているが，ウィンドウ枠を拡大すれば10本の列車まで表示できる．そして発車時刻1分前になると表示が点滅し，発車が近づいたことを知らせる．そして発車時刻を過ぎると，その列車は案内板から消えて，次発以降の列車が1段上に移動する．またオプションで，今度発車する列車についてのより詳しい案内を表示することができる．字幕はウィンドウの最下段に右から左へ水平スクロールしながら表示される．ウィンドウサイズの調整で10本の列車と字幕をすべて表示することができ，またTRAINSを複数起動すれば大きな駅でのそれのように多くの線区の案内板を並べて表示できる．

起動

WTRN303.EXEはディレクトリごと圧縮された自己復元型圧縮ファイルとなっているので復元時はMS-DOSのコマンドラインから

```
A> WTRN303
```

と入力すれば，カレントディレクトリにデータファイル用のディレクトリを作成し，それぞれのサブディレクトリにファイルを復元する．

次に通常のWindowsアプリケーションと同様に「TRAINS.EXE」を起動する．TRAINSを起動するとダイアログボックスが開くので，案内を表示したい列車のデータの入った「列車情報ファイル」を選択する．

Windowsのプログラムマネージャに TRAINSを登録する場合，登録コマンドラインのオプションに列車情報ファイルの存在するディレクトリを指定しておくと，起動後，ダイアログボックスが開いたときに最初からそのディレクトリを表示することができる．

また，登録コマンドラインのオプションに列車情報ファイルをフルパス名で指定すれば，起動後ダイアログボックスを開かずに，ただちに案内板を表示することができる．

さらに列車情報ファイルをTRAINSに関連付けすることで，列車情報ファイルを指定すればTRAINSを起動することができるようになる．関連付けは，Windows初期化ファイル「WIN.INI」の［Extensions］セクションに

```
trn=trains.exe ^.trn
```

と追加すればよい．

起動後，表示している列車を変更したい場合はコントロールメニュー内の「案内選択...」コマンドを実行する．このコマンドは，アクセラレータキー([CTRL]+[B])で呼び出すこともできる．

またコントロールメニュー内の「字幕表示」コマンドを実行すると，ウィンドウの最下段において字幕表示を行うことができる．このコマンドはトグル動作で，実行するごとに表示ON／表示OFFが切り替わる．ただし字幕表示を行うためには，現在表示している列車情報ファイルに字幕情報が含まれている必要がある．字幕情報は個々の列車情報ファイルによって含まれていたり，そうでなかったりする．字幕情報が含まれていない場合，字幕は表示されない．このコマンドも，アクセラレータキー ([CTRL]+ [D]) が利用できる．

また，マウスポインタがウィンドウ内にある（マウスポインタが切符の形

ワンラインNEWS
米インテルは 91 年半導体生産額で米モトローラを逆転し，首位になったと発表した．世界ランキングでは日本電気，東芝，日立製作所の上位3社に変化はない．

状になっている）とき，マウスの左ボタンをダブルクリックしても変更することができる．

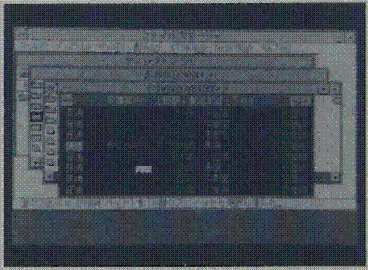
コントロールメニュー内の「発車案内について...」コマンドを実行すると，ダイアログボックスが開き，その下に現在オープンしている列車情報ファイルのデータが表示される．このコマンドは，アクセラレータキー（[CTRL]＋[A]）で呼び出すことができる．

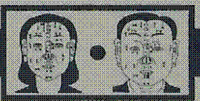
最寄りの駅の案内板が作れる

列車情報ファイルは拡張子に「.TRN」をもち，Windows初期化ファイル「WIN.INI」と似た書式をもったテキストファイルである．個々の列車情報ファイルには，ある駅（例：上野駅）の，ある線区（例：常磐線）の下り，または上り線の1日分（始発から最終まで）の列車のデータが格納されている．紙面の関係からその内容は割愛するが，ドキュメントファイルに詳しい説明があるので，時刻表を片手に作ってはいかがだろう．またTRAINSの列車情報ファイルのデータが，有志の手によって一部の線区ではあるがNIFTY-Serveなどにアップロードされている．

■TRAINS Version 3.03

名称：TRAINS／分類：列車発車時刻データベース／登録名：WTRN303.EXE／作者：南川宏（NIFTY-Serve HCB00054）／動作環境：MS-Windows 3.0が稼働するマシン／登録月日：1991年12月20日／復元方法：WTRN303／入手場所：NIFTY-Serve FFMPRO DL #4



HEALTHY?  **UNHEALTHY?**

⑮ 電磁波の謎③

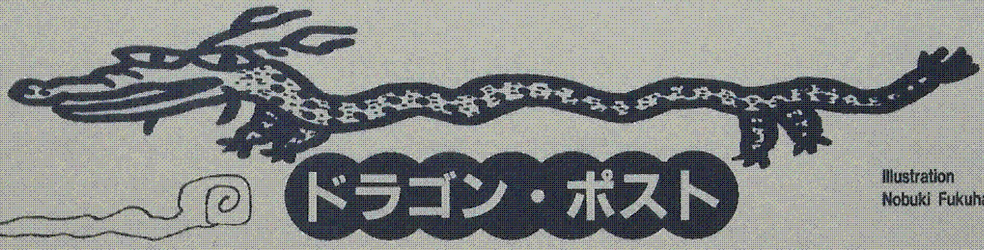
CRTディスプレイから目や身体を守るためのフィルターがあるということからはじまって，CRTディスプレイの電磁波のことを少し調べてみたのだが，どうも電磁波の害というのははっきりしない．高電圧で電子ビームを発射するときにX線が発生する危険はあるらしいが，現実にそんなものが製品として出ているわけがない．むしろ，X線が発生する可能性からくる不安が電磁波の害に敏感になる心理をうんでいるのかもしれない．

実際に目への影響ということを考えると，グレアやフリッカのほうが問題になる．グレアは，まぶしさを意味し，おもにCRTディスプレイに反射する光源が問題となる．窓を背にしてディスプレイと向き合えばよくわかるが，ディスプレイに外光の映り込みがあると見づらいことこのうえない．蛍光灯など照明の反射も同じである．また，反射だけでなく，たとえばCRTディスプレイの向こう側に照明などがあって，それがときどき目にはいるという場合でも，目への影響がある．

こうしたことは環境さえ許せば，デスクやライトの位置を考えることによって変更することが可能である．どうしてもデスクの位置が動かせないなどの事情によって，映り込みがある場合は，ディスプレイ用のフィルターが役に立つ．フィルターは，電磁波よりもノングレアのために使うのが現実的だろう．

画面全体の明るさよりもかなり明るいものが視野に入っているとグレアの原因となる．一太郎のようなアプリケーションの場合，画面の背景が黒だから，映り込みの影響も受けやすい．一方，Windowsのような画面では，全体が明るい色彩で表示されることが多いから，周りが少し明るくてもまぶしさをあまり感じないだろう．もっとも，Windowsの色調はユーザーが調整できるから一概にはいえない．

画面が全体に明るいと，今度はフリッカ（ちらつき）が気になることがある．画面が明るいほどちらつきは目だつ．ちらつきがディスプレイの不調によるものなら修理したほうがいいし，ディスプレイを買うときも，できるだけ実際に使うソフトに近い状態でディスプレイを見て選びたいものだ．



ドラゴン・ポスト

Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成しています．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，はがきには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★Cがわからないと……

大学の研究室に配属されるなり，「Cがわからないと，研究できないぞ」と，おどかされ，2年前にパソコンを購入．でも，研究室の誰もCをわからないなんて……．雑誌や本を参考にして，独学でなんとかなるまでになる頃には，世の中は，C++やWindowsへ．だんだん雑誌が遠ざかって行くように思います．(EG)

――基本的な技術をしっかりと身に付けておけば，応用はそれほどむずかしくないはずです．周りに振り回されずにキチンとプログラミングを学びましょうね．

★そのご意見に賛成！

ドラゴン・ポスト（1月号）の「Cだ，Cだというけれど……」に賛成．近頃のCの流行には疑問を感じる．確かに書き易いという面もあるのだろうけれども，ちょっとマニアックな感じがする．初心者には不向きであろう．もっといろいろな言語が乱立しても良いのではなかろうか．(橘内正人)

――プログラミング言語は星の数ほどもありますし，それぞれ得意な分野をもっています．初心者には，やはりPascalなどがいいのでは？

★FP-1100健在！

1月号のパソコン博物館の「FP-1100」を持っています．FP-友の会というのがあって，現在180人位の会員がいます．会報は月刊えふぴィといって，これは106号まででています．もちろん私も会員です．(庄司孝志)

――古いパソコンというと，私もApple IIcとFM-11AD2+をもっています．ほとんど動かすことはないのですが，なぜか捨てられません．^^;

★C言語との戦い

ソフトウェアの仕事について7年経ちます．その間，某社の汎用コンピュータを使用し，事務処理系のソフトばかりを作成してきました．当然，C言語などとは縁がなく，COBOLオンリーの世界です．そこで！ 28才という高齢(?)にもかかわらず，現在“C言語”と必死で戦っております．はてさていったいいつになったらものになるのかな？(増田雅彦)

――COBOLを身に付けているのなら，Cでもすぐプログラミングできるようになりますよ．でも，COBOLライクなCプログラムになってCプログラマの顰蹙をかうかも？

★アイデアはどんな時に……

今，私は胆石摘出手術を終えて，病院のベッドで静養している．自分の時間は充分にあるのだが，なかなか良いアイデアが湧いてこない．なんとまあ，もったいない話ではある．入院前は，パソコンの前に座ると，アイデアが頭の中をヒコーキ雲のように横切っていったのだがなア……．アイデアはどんな時に浮かんでくるものだろうか？(安永誠治)

――編集者がもっともおもしろいアイデアを出すのは，飲み会の席ときまっています．ただ，実現不可能なものが多いというのが玉に傷ですね．酔っぱらいのタワゴトという話もあるし……．

★USに負けないアプリを作ろう！

Windowsのアプリを書いています！ これからのOS，Windowsなのですが……，社内環境のネットにつなげると，EMSの番地がぶつかって動かなくなったり，etc！とにかく，全てはこれからだと思います．U.S.A.のソフトに負けないくらいのアプリをつくりましょう！皆さん．では，また……．(沢田秀幸)

――PascalやBASICの処理系も登場してWindowsプログラミングもずっと身近になりました．LHAのように本場アメリカへ逆上陸するソフトが現れてほしいものです．

★サポート用にプレゼントが欲しい！

会社でC言語のお客様サポートをやっているのだが，これまでのPC-9801のみから，IBMや富士通，東芝などにも対応するという．サポート担当にも全機種そろえてくれるのだろうか？ それが心配で読者プレゼントに応募してみる，おちゃめな私．ナンテネ．(今井文一郎)

――なんだかずいぶん不安になるサポートですねぇ．大丈夫なんですか？ 「いやぁ，当社にはそのような機種はありませんからわかりませんね．」とか(笑)．

★便利ツールに慣れすぎると
以前から便利ツールの存在は知っていました．また，いくつかは使ってみたこともあります．しかし……，普段からツールを使って仕事（プログラミング～テスト）をしている環境になれてしまうと，思わぬ落とし穴にはまると思います．それは，例えば納品後のトラブルでリリース先へ行って修正作業を行わねばならないような時です．場合によってはEDLINしかない場合だって，あるいは一太郎しかない時もあります．その時は，手も足も出ない．相手は何らかの改善がなければ帰してくれない雰囲気．あせりますよ……．だからボクとしてはツールは最低限しか使わなくなりました．あんがい，FINDやEDLINだって使い方では有用なものだと思いますが，どうですか？（佐野浩之）
——貧弱な環境でのプログラミングに耐えて自分を鍛えるのもいいですが，開発に使えるノートサイズパソコンの登場を祈るというのはダメですか？　軟弱……？^^;

★世の中，変だ！
家には386（20MHz）のマシンに4MもRAMを拡張して（今時は4Mしかかな？），HDも180Mもつけているのに，大したことはやっていない．バイト先ではV30だの286だので，コンパイルする環境でプログラミングしている．世の中，変だ．（HDDはGAME用？）
——世の中そんなものですよ．プログラマーズ・ページ編集部だって……シクシク;-（

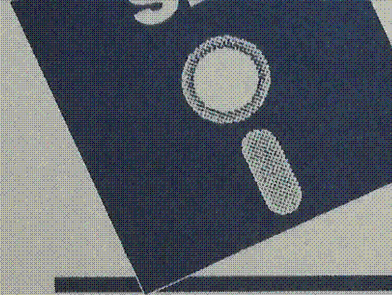
★全治３か月
右手人指し指の腱を切ってしまった．ただの突き指だと思ったら，指がまったく動かないので病院に行ったところ，そうであったとわかりました．が……．oh，No!　キーボードが打てないよー．字も書きにくい．このせいで，パソコンに向かう時間が短くなった．その分，本でも読みましょう．ツライ……．（全治３か月だって……）
——災い転じて福と成す，というわけでワンハンドタッチタイピングができるキーボードを考案してみるってのはどうでしょう？

★りえちゃんのヌードは……
私は大学の電子計算機センターでアルバイトをしています．ここにはWSが70台，FM-R70が110台ぐらいあります．ある日，WSが並んでいる部屋を掃除していた時，「俺は札束に囲まれているようなものだなあ．」と感動にひたっていた．しかし，ふと顔を上げると，そこには宮沢りえのFM-TOWNSのポスターが!!　「WS70台よりこいつのヌードの方が高いんだろうなー」と思ったとたん，さっきの感動はどこかへ飛んでいってしまいました．すごくむなしい気分になりました．（電大万歳！）
——そりゃあなた，コンピュータなんて機械よりも若い女の子の体のほうが価値があるにきまってるじゃないですか．某社のディスプレイからはいつもりえちゃんがプログラマに優しく微笑んでいるし……．

★頭が痛い
ここ１年位のパソコン界の環境の変化は，ほとんどついて行けない位でした．原因はWIN3と，そしてDOS/Vでしょうね．プログラマの私としては，このどちらも放っておけませんし，また，WIN3ではOOPという世界も理解しないと，今後苦しくなってしまいそうです．通り一遍の処理で済んでいた汎用機の世界も徐々に減退して行きそうですし，ダウンサイジングの合言葉の元に，小型機のLAN構築が必要不可欠なものとなってしまいました．あ，頭が痛い……．（小島伸一）
——でもこれはチャンスでもあるんですよ．Windowsプログラマなら月100万程度の給料を払うという会社はザラですから（というウワサですけどね）．

★どんな人が作っているのか……
たとえば，銀行のCD機用のソフト，車のナビゲーション，ビデオの予約など，機械に組み込まれている物は，いったいどんな会社のどんな人々が，どんな環境で作っているのか知りたいものだ．（古田光寛）
——テレビ番組でパチスロのROM改造を専門にしている“裏ROM師”なる人物を見ましたが，やっぱり開発は98でやっていましたね．symdebも使ってましたねぇ（笑）．

★X68000はすごいよ！
X68000ユーザーの僕としては，大分ものたりない．まあ，他の本でもわからないから，内容がある分よいかな．X68000は使わないのに，あんなのだめだ！　という人が多くて．BIOSがソフトウェアで提供されるから，初期の５倍位の速度になっているんだよ！（BIOSが）．SX WINDOW（X68000のウィンドウシステム）も，MS-WINDOWSよりずっとよいと思う．やっぱGUIはセンスがなくちゃ！　68000MPUの魅力，16Mリニアをもっと知ってもらいたい．すんごいよ．（じゃがいも）
——X68000には，懐かしのOS-9がありますね．X68000用のUNIXを出せばいいのに．こんなこと考えるのは私だけかな？



付録ディスクのご案内

このフロッピーディスクに含まれる実行ファイル，ソースリストおよびテキストファイルは「月刊Programmer's Page」1992年3月号に掲載されたものです．

付録のフロッピーディスクは，5"2HD（8セクタ／トラックMS-DOS）でフォーマットされています．3.5"や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5276-1297）でも，同じ内容のファイルを用意しています．

フロッピーディスクの内容は次の通りです．本誌およびディスクに含まれるドキュメントファイルをよく読んでからご利用ください．

■ディスクのおもな内容

●付録ディスクについての説明
ディレクトリ名：ルート
収録ファイル名：
README.BAT（ディスク内容の説明を読むためのバッチファイル，PC-9801専用）
README.DOC（ディスク内容の説明，MS-DOSテキストファイル）
ONLINE.BAT（オンラインソフトの説明を読むためのバッチファイル，PC-9801専用）
ONLINE.DOC（収録されているオンラインソフトの簡単な説明）
V.EXE（ミニページャー，PC-9801専用）
BOOK.TXT（翔泳社書籍目録）

●特集「Cじゃなくてもいいじゃない」
①「Pascal vs C」(p.13)
ディレクトリ名：TOKU
収録ファイル名：TOKU1.LZH
②「MODULA-2 vs C」(p.24)
ディレクトリ名：TOKU
収録ファイル名：TOKU2.LZH
③「Lisp vs C」(p.40)
④「Prolog vs C」(p.56)
⑤「SmallTalk vs C」(p.64)
ディレクトリ名：TOKU
収録ファイル名：TOKU3.LZH

●プログラム「遊」工房(p.80)
簡易バイナリエディタ「bedit」
ディレクトリ名：PLAY
収録ファイル名：PLAY.LZH

●イカすBASICプログラム入門(p.91)
謎のプロシージャ編
ディレクトリ名：BASIC
収録ファイル名：QBASIC.LZH

●実践C塾(p.98)
サーチその1
ディレクトリ名：JUKU
収録ファイル名：JUKU.LZH

●98活用アセンブラ技法(p.110)
ディスクイメージダンププログラムの作成
ディレクトリ名：98ASM
収録ファイル名：98ASM.LZH

●綴じ込み付録
ディレクトリ名：DATA
収録ファイル名：DOSFUNC5.TXT（MS-DOSテキストファイル）

●オンラインソフト採集帳(p.144)
本誌今月号のオンラインソフト採集帳で紹介した列車発車時刻案内板「TRAINS」と高圧縮復元書庫管理ソフト「LHA」が収録されています．拡張子が「.LZH」のファイルの復元方法についてはLHAに含まれるドキュメントを，LHAの簡単な使用方法についてはONLINE.DOCを参照してください．
ディレクトリ名：ONLINE
収録ファイル名：
WTRN303.EXE（TRAINS，MS-Windows 3.0用）
LHA213.EXE（LHA v.2.13，MS-DOS汎用）

●クイズ・ランダムスクリーン(p.150)
クイズ解読ツール「PUZLE」
本誌掲載のクイズ・ランダムスクリーンの画像をディスプレイ上に再現し，マウスを使って並べ換えることにより，解答を得ることができるユーティリティです．PC-9801シリーズ専用です，詳しい使い方はPUZLE.DOCを参照してください．
ディレクトリ名：OMAKE
収録ファイル名：
PUZLE.EXE（解読ツール，PC-9801専用）
PUZLE.DOC（PUZLEの説明）
CONFIG.SYS（PUZLE用CONFIG.SYS）
AUTOEXEC.BAT（PUZLE用AUTOEXEC.BAT）
RNDSCREN.LZH（画像データ）
MIKAN.LZH（2月号の解答データ）
MIKAN.BMP（Windows 3.0の壁紙）

※使用上の注意

本誌に掲載されたプログラムおよびデータを利用するには，MS-DOSの基礎知識が必要です．このディスクには，MS-DOSのシステムは入っていません．別途，MS-DOSが必要となります．また，ソースファイルのコンパイルなど，別途コンパイラやユーティリティが必要な場合があります．

■圧縮ファイルについて

ディスクスペースを有効に使うために，多くのファイルは圧縮した上で収録しています．「.LZH」という拡張子のついたファイルは，高圧縮書庫管理ユーティリティ「LHA」によって圧縮されたファイルです．これらのファイルを利用するには，まずLHAで復元する必要があります．LHAの利用方法については，ONLINE.DOCをご覧ください．

バーチャルスペースガイド

C平　1月の12日にHALのバースデーパーティーが開かれたんだってね。もちろんアメリカでのことだけど。

P子　HALって，「2001年宇宙の旅」に出てたコンピュータのこと？

C平　そうあのHAL。1992年の1月12日に誕生したことになっている。その0歳のお祝いなんだって。

P子　あの映画，不思議にリアリティがあって，意味がわかんなくても見てしまう魅力があるよね。でも，ずいぶん前の映画じゃない。

C平　何と24年前の映画。僕たちは，あの映画ができたときには「未来」だった時代に住んでいるわけだけど，まだ，あの世界は「未来」のままだね。知能を持ったコンピュータなんて，ずーっと先の話だと思うもの。

P子　一時，人工知能とかニューロコンピュータとかが話題になったけど，いまニューロっていったら，エアコンか洗濯機のことかと思ってしまう。

C平　でも，ニューロの研究はしっかりやっているみたいだよ。ニューロ基盤を作ったり，人間の脳の研究を反映させたりして，いまのコンピュータの学習機能っていうのは，何回も同じ操作をしているうちに，確率の高い答を出せるようになるものじゃない。人工知能というのは，コンピュータ自体が判断して情報を組織するものになるらしいんだ。ニューロの専門家も意識を持ったコンピュータを作るには，50年や60年はかかるんじゃないかといっている。

P子　あー，そのころには，私いくつになるんだろう。

■2001年宇宙の旅

(1968年，米)
監督：スタンリー・キューブリック
出演：キア・デュリア

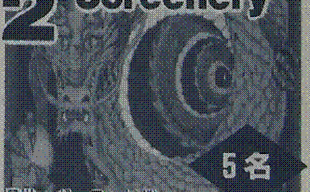
SF界の大御所アーサー・C・クラーク原作によるSF映画の古典的名作。人類発祥と宇宙の存在に関わる壮大な構想を持って作られた映画であるが，意味不明という声も少なくない。映画のテーマの謎と丁寧な宇宙旅行の描写が評判を呼び，20年以上を経た現在でもファンが多い。知能を持ったコンピュータHALの反乱が物語の重要な部分を占めている。

読者プレゼント

1 LOGITECH MODULA-2 VERSION 3.0 [*1]

1名
提供：㈱NJK―

2 Screenery [*2]

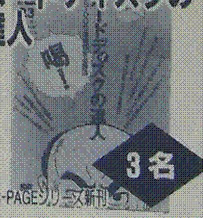


5名
提供：ボーランド㈱

3 MS-DOS 5.0 全ファイル解説

3名
提供：翔泳社・PC-PAGEシリーズ新刊

4 ハードディスクの達人



3名
提供：翔泳社・PC-PAGEシリーズ新刊

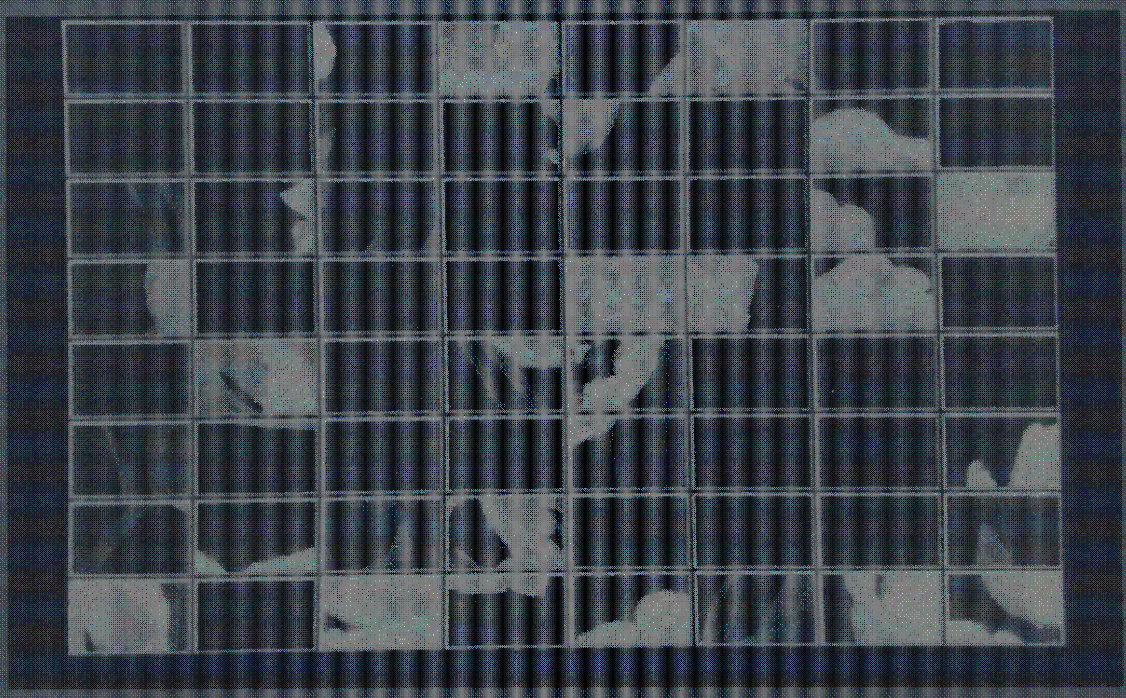
応募は本誌に添付のアンケートはがきをご利用ください。(1992年3月17日消印有効)。なお，応募者多数の場合は，抽選になります。当選者発表は発送をもって代えさせていただきます。

＊1：PC-9800シリーズ用，3.5インチ
＊2：MS-Windows 3.0対応

クイズ・ランダムスクリーン

ある1枚の画像を64分割し，ランダムに並べ替えてあります．
さて，もとの画像は何を写したものでしょうか．

ヒント：咲いた，咲いた……



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます，

応募方法：本誌に添付のアンケートはがきをご利用ください．

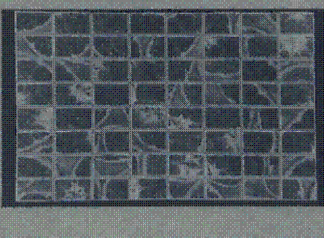
締め切り：1992年3月17日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

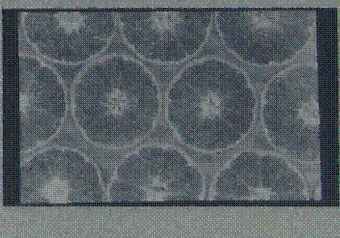
＊画像データは，付録ディスクやBBSでもサービスしています．

2月号の問題と正解

【問題】



【正解：みかん】



SE-NETからのお知らせ

SE-NETでは開設以来，オンラインサービスを行ってきましたが，回線混雑のため非常に接続しにくい等の問題が出てまいりましたので，GUESTによるダウンロードは中止させていただきます．今後は読者のみなさまと編集部とのコミュニケーションの場としてご利用ください．

●BBSへの接続

SE-NET　03-5276-1297

通信速度	300/1200/2400bps
接続プロトコル	CCITT V.22bis/V.22/V.21
エラー訂正プロトコル	CCITT V.42，MNPクラス5
漢字コード	シフトJIS

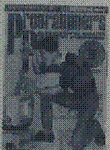
Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ バックナンバーのご案内

■定価・各980円
（1991年１月号「特集・プログラマの仕事場」・２月号「特集・Windowsの時代である!」・３月号「特集・この言語を使いなさい!!」・４月号「特集・Ｃ読本」は品切れです）

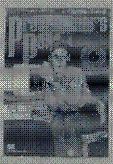
1991年5月号 5インチFD付き
特集・基礎力としてのアセンブラ

プリミティブなプログラミング言語でありながら，それ故に無視することができないアセンブラ，コンピュータを理解し，その能力をひきだすためにもう一度アセンブラをみなおす。



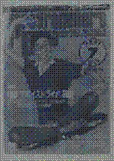
1991年6月号 5インチFD付き
特集・研究「プログラミング統合環境」

今流行のエディタ，コンパイラ，リンカ，デバッガを統合した開発環境について，その歴史的系譜から最新のシステムまで徹底的に検証する。



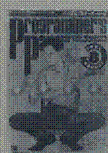
1991年7月号 5インチFD付き
特集・最新「デバイスドライバ」研究

パソコンの拡張性を支えるソフトウェアインターフェースであるデバイスドライバ作成の技法と特徴をMS-DOS, Windows, OS/2の各環境ごとに解説する。



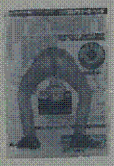
1991年8月号 5インチFD付き
特集・はじめてのオブジェクト指向プログラミング

“オブジェクト指向”の歴史的な発展の過程から基礎概念，分析・設計の方法までを網羅的に解説する。



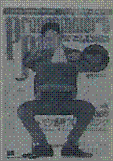
1991年9月号 5インチFD付き
特集・グラフィックスプログラミングの世界

基礎からフレームバッファの活用，レイトレーシング，Windows，IBM PCのグラフィックスなど，その技法を徹底的に解説する。



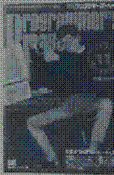
1991年10月号 5インチFD付き
特集・286/386メモリ活用プログラミング

バンク式メモリ，EMS，XMSからプロテクトモードまで，640KBの壁を破るための数多くのプログラミングテクニックを解説する。



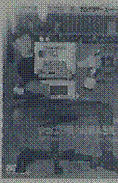
1991年11月号 5インチFD付き
特集・Windowsプログラムの作り方

正統派MS-C＋SDKをはじめ，VisualBASIC，BC＋＋，TPWなどの新しい開発環境，ツールによるWindowsプログラムの作り方を探る。



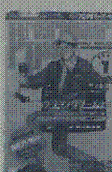
1991年12月号 5インチFD付き
特集・やっぱり僕らはBASIC!

構造化プログラミング，マルチランゲージに対応した強力なプログラミング言語として生まれ変わった現在の進んだBASIC，MS-BASIC 7.1，TrueBASICからVisualBasicまでを紹介する。



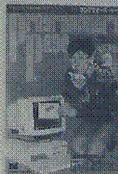
1992年1月号 5インチFD付き
特集・プログラミングツール大集合

プログラム開発，メンテナンスに欠くことのできないプログラミングツール，その利用法から，使いこなしのコツまでを紹介する。



1992年2月号 5インチFD付き
特集・通信ソフトの作り方

RS-232Cを使用した通信の概要を解説し，ライブラリ，ライブラリを用いた通信用マクロ言語を作成，BBSなどにアクセス可能な通信プログラムを完成する。



「月刊プログラマーズ・ページ」のバックナンバーは，ご注文いただければ，全国どこの書店でもお求めになれます（書店に届くまで１か月ほどかかります）。
小社に直接ご注文なさるときは，何月号かを明記の上，現金書留にて雑誌価格＋送料を下記までお送りください。送料は１部81円です（ただし７月号は１部86円）。２冊以上の場合は，号数・冊数により送料が異なりますので，小社までお問い合わせください。

〒102 東京都千代田区平河町2-4-14
平河町KSビル4階
（株）翔泳社
マーケティング部PP係宛
TEL 03-3263-0447

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ4月号
（3月18日発売）

付録：5″ 2HDフロッピーディスク付き!!

特集

新しいDOS環境を探検する

MS-DOS 5.0，DOS/Vプログラミングの知識と技術

数々の新機能を備えたMS-DOS 5.0やIBM PC/AT上で日本語の使用を可能にしたDOS/Vの登場により，新しいプログラミング技術が必要とされている．バージョンアップしたMASM 6.0の情報も加えて，新しいDOS上でのプログラミングを探る．

月刊プログラマーズ・ページ　1992年3月号
1992年3月1日発行

発行所　株式会社翔泳社
東京都千代田区平河町2-4-14平河町KSビル4F・5F
TEL 03-3263-0447　〒102
発行人　長廻健太郎
編集長　清水　隆
副編集長　鈴木嘉平
編集スタッフ　杉野貴美子＋上田純美礼＋芳山充弘
表紙デザイン　SE（翔泳社デザイン研究所）
広告スタッフ　相澤正夫＋澤本良知＋種村嘉彦
協力　根本裕三郎＋(株)ドモン・マインズ＋佐藤裕美＋保坂庸介＋小林弘人(図版デザイン)
写植　株式会社東巧社＋株式会社トーヨー・プリント社
印刷　凸版印刷株式会社
定価　980円（本体951円，税29円）

© 1992 SHOEISHA
本書の一部または全部を著作権法の定める範囲をこえ，無断で複写，転載，テープ化，データファイル化することを禁じます．
＊本書の内容に関するご質問は，ご面倒でも必ず文書にて編集部までお問い合わせください．
＊本誌に記載した会社名，ハードウェア・ソフトウェアの商品名は，各社の商標または登録商標です．

EDITOR'S PAGE

●読者から寄せられたハガキに「Windowsは楽しい」というものがあった．環境設定や画面設定をユーザー自身でできるから，確かに楽しいソフトウェアである．CONFIG.SYSを書き換えたり，コントロールパネルをいじったり……，こうしたことを楽しいと感じる人は少なくない．一方で，それを「面倒だ」という人もいる．多分いまのパソコンは，「楽しい」と感じる人のほうが馴染みやすいものに違いない．楽しみながらさまざまな知識を身につけていった人が多い．しかし，今後ソフトやハードがどんどん使いやすくなっていくに従って，そうした楽しみは奪われてしまうのかもしれない．(TA)

●486マシンがついに我が家にやってきた．486DX33MHz，8MBメモリ，535MBHDD，3.5″FDD×1，5.25″FDD×1，17″ディスプレイ．う～ん，広いWindowsはいいなぁ．1024×768×256色で，○えちゃんが微笑んでるし．あとは，サウンドブラスターとマルチメディアエクステンションを手に入れて，もっと高速なグラフィックアクセラレータボードと入れ換えて，そうだCD－ROMも買わないといけないし……，あぁ，金が，金が……．(嘉平)

●冬休みに香港へ行き，噂の「黄金購物中心（ゴールデン・ショッピングセンター）」も探検してきました．迷路のようなビルの中に，PCのハード，ソフト，ファミコンなどの電脳（コンピュータ）がひしめきあい，地元のパソコン少年や欧米人で賑わっていました．ほとんど観光客が立ち入らない地域にあるため，気弱な（？）私には少し恐ろしげでしたが，ジュースやラーメン（たぶん）を売る屋台が表に並び，香港の雰囲気満点．物価の安さや料理の美味しさもさることながら，香港人の強力なパワーに圧倒され，感化されてきました．元気になる香港にまた行きたい／（ま）

●アプリケーションソフトのMS-DOS 5.0への対応には少々不満があります．対応しているかどうかをユーザーが自分で確認しなくてはならないなんて，どこかおかしい．本来，こういうときの用心のために行っているはずのバージョンチェックを，SETVERコマンドで回避するよう指示するなんて，対応とはいえないのではないでしょうか．それでも「対応しない」という対応よりは反応があるだけまだまし？（kos）

●友人が初滑りで，しかも，たった数百メートル滑っただけで今シーズンを棒にふった．骨折で入院．猪苗代リゾートスキー場も，オープン日の午前中にいきなり怪我人がでるなんて考えもしなかっただろう．他人事ではないかもしれない．ところで，僕をこの世界に引き込んだ悪しき友人が，この号が発売される3日ほど前に結婚する．幼女誘拐殺人のM君に瓜二つの彼も，（あまり期待はできないだろうが）これでまっとうな人生に戻れるかもしれない．倉橋君，おめでとう．(yoshi)

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVERY CARD　愛読者カード

本誌の号数　　年　　月号

本誌を知った理由
- □1.書店店頭　□2.他社出版物(　　　)
- □3.小社出版物　□4. DM　□5.口コミ

パソコンの所有形態(機種名)
- □1.個人所有　(　　　)
- □2.職場にある(　　　)
- □3.持っていない

本誌を購入した書店　　市(区)　　書店

よく使用するソフトウェア

よく使用するOS

今月号でおもしろかった記事(いくつでも)

今後取り上げてもらいたい特集のテーマ

貴方が選ぶベスト3

パソコン雑誌	①	②	③
ソフトウェア	①	②	③
パソコン本体	①	②	③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください(「ドラゴン・ポスト＝読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください)。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社(学校)名		9	所属(学部)		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

業種　①製造業　②金融・保険・不動産業　③商社・卸・小売業
④建設業　⑤通信・運輸　⑥システム・ソフト
⑦その他サービス業　⑧公務員　⑨その他

職種　①コンピュータ(ハード)技術者　②コンピュータ(ソフト)技術者　③エレクトロニクス技術者
④プログラマ　⑤その他の技術者　⑥専門職
⑦事務・営業職　⑧経営管理職　⑨商工サービス
⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　　年　　月号

希望項目 ＼ 広告資料請求No.												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊＝11,760円（本体11,417円/税343円）
郵送料サービス

郵便はがき

料金受取人払
麹町局承認
3908

差出有効期間
平成4年11月
30日まで

切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
マーケティング部 行

月刊プログラマーズ・ページ
READER'S SURVEY CARD　愛読者カード

氏名	性別 □1男 □2女	生年月日 19 年 月 日
住所 〒		電話 （ ） －

職業
□1ソフト技術者 □2ハード技術者 □3技術系会社員
□4事務系会社員 □5営業系会社員 □6教師
□7一般公務員 □8経営管理職 □9理系学生
□10文系学生 □11その他（ ）

ユーザー区分
□1プロ
□2マニア
□3一般ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

郵便はがき

料金受取人払
麹町局承認
3910

差出有効期間
平成4年11月
30日まで

切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

＊当社で記入

1	SE	2	受付No.	

月刊プログラマーズ・ページ広告資料請求カード

3	会員番号	

＊すでに会員番号をお持ちの方はご記入ください。

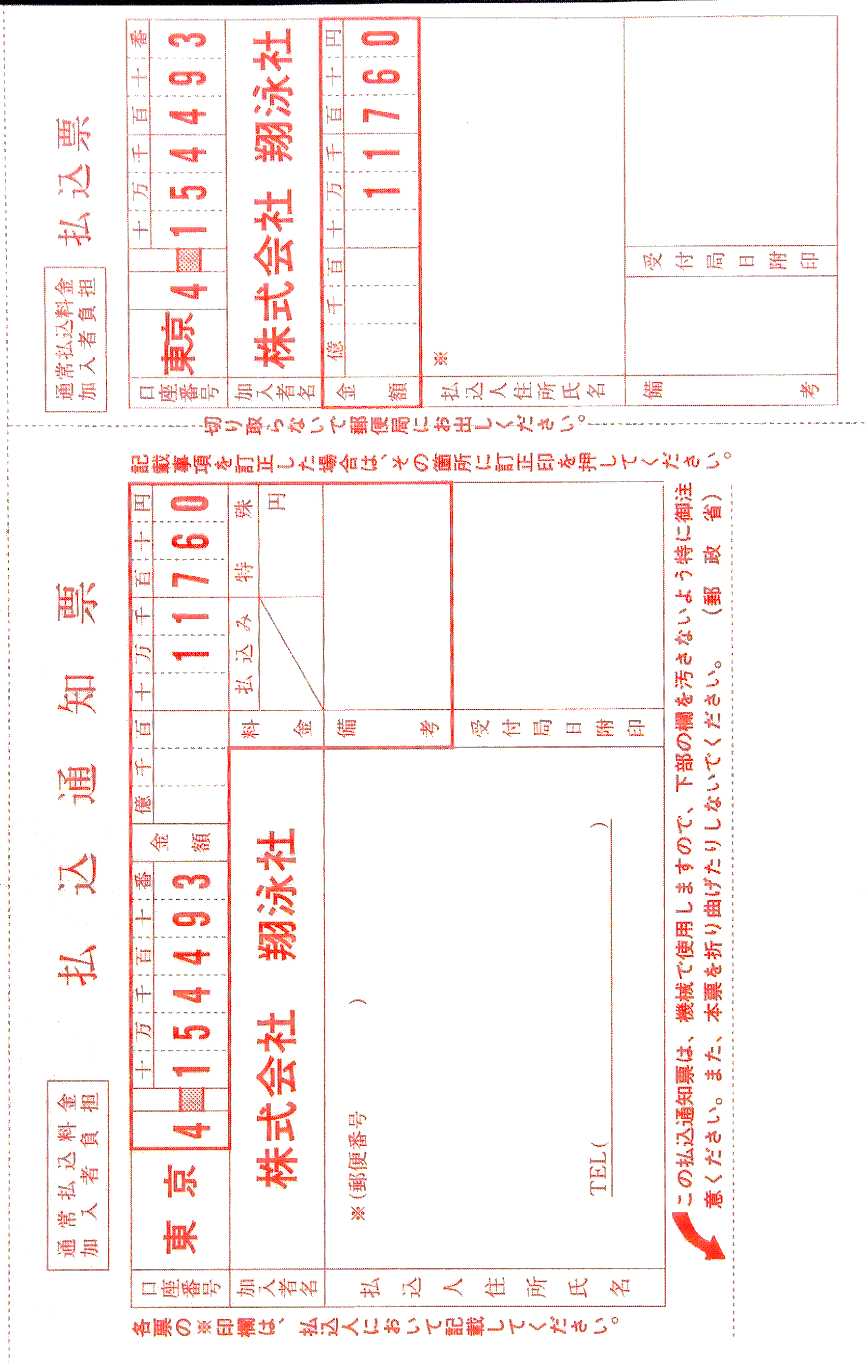
4	フリガナ 氏名	姓　　名	5	生年月日	19 年 月 日

6	自宅住所	〒　電話（ ）
7	会社住所	〒　電話（ ）

資料送付希望先の番号に○印をつけてください。

…ビス株式会社

…、国内外のプラント建設を手
…(株)の100%出資子会社で、
…社。東洋エンジニアリングの
…ちろん、一般企業へのプログ
…リなどの人材派遣、受託のプ
…等の販売代理と、独自の営業
その中で、海外の優れたソフ
…、日本のマーケットに提供す
…り組みを見せている。いま話
…含む32ビット開発ツールキッ
…同社から発売されたものだ。
…にシステム営業部の松本孝之
…をうかがった。



出てきた。これらは非常に優れた開発環境を提供するものですから、日本にもぜひ紹介したい、根付かせたいと考えました。

海外のソフトを日本のマーケットに提供する場合に、コンセプトもなくワープロからゲームまで何でも、という考えは持っていないんです。やはり、日本のコンピュータ環境にとって意味の高いもの、新しい価値を付け加えられるようなものを供給していくことが大切なんだと思います。とくに開発関シンへの移植をメーカーにやらせるのではなく当社でやる。マニュアルも当社で翻訳して出す。またテクニカルサポートも右から左へメーカーに流すのではなく、できるだけ当社で対処する、ということです。当社独自の付加価値を付けて出すということで、当然メーカーから出しているものとは多少違うところが生まれてきます。そこに私たちのやりがいもあるわけですし、また当社の技術力を上げていく力にもなっていくんだろうと考えています。」

日米の文化の違いを乗り越えるのにひと苦労

…、海外のソフトを日本語化する
…、日本のパソコン環境の特殊性
さまざまな問題がつきまとう。
…のメーカーと仕事をしていく中
…るのは、まず文化の違いですね。
…商業常識とアメリカの商業常識
…ますから、日本では当然やるよ
…ともアメリカではやらないとい
…とがあります。またアメリカで
…ことも日本ではやらないという
…あります。一番苦労しているのは、ライセンス契約でしょうか。日本におけるライセンス契約とぜんぜん違いますので、日本のお客さまに違和感を感じられる部分で、なんとか納得していただくのに苦労することがあります」と松本氏。実際にどのような点が問題となるのか、さらにうかがってみた。

「いままでやってきて驚いたのは、たとえば契約書に〝神に誓いなさい〟とか〝不慮の事故は神のみぞ知る事柄なので、責任はもちません〟とか、そういう条項が出てくることですね。

定期購読のご案内

■定期購読は、1年間12冊と限定させていただきます。
■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。
■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。
■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。
■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

この欄は、加入者あての通信にお使いください。

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760 円（税・送料込み）

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵政省）

郵便はがき

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

2 受付No.

マーズ・ページ広告資料請求カード

＊すでに会員番号をお持ちの方はご記入ください。

名	5	生年月日	19　年　月　日

□10文系学生　□11その他(　　　)　ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

6	自宅住所	〒　電話（　）
7	会社住所	〒　電話（　）

資料送付希望先の番号に○印をつけてください。

綴じ込みの 1

テックソフト&サービス株式会社

ちょっと気になるこの会社

広告企画

テックソフト&サービスは、国内外のプラント建設を手掛ける東洋エンジニアリング(株)の100%出資子会社で、設立して5年目という若い会社。東洋エンジニアリングのコンピュータ部門の管理はもちろん、一般企業へのプログラマやオペレータ、セクレタリなどの人材派遣、受託のプログラム開発、ハードウェア等の販売代理と、独自の営業活動を幅広く展開している。その中で、海外の優れたソフトウェアを輸入、日本語化し、日本のマーケットに提供するという業務にも意欲的な取り組みを見せている。いま話題のDOS-Extenderを含む32ビット開発ツールキット『386/TOOLBOX』も同社から発売されたものだ。千葉県船橋市の本社オフィスにシステム営業部の松本孝之氏、菅原裕之氏を訪ね、お話をうかがった。

開発環境中心に米国ソフトを日本語化

同社の提供するソフトウェアには、『386/TOOLBOX』を始めとして、『HIGH C386』、『NDP FORTRAN 386』など開発関係が目立っている。その狙いがどの辺にあるのか、まず松本孝之氏に語っていただこう。

「システム営業部では3年ほど前までは、ファイルコンバータやデータベースなど、日本国内で開発された商品を提供してきました。それがアメリカでDOS-Extenderなどの新しい商品が出てきた。これらは非常に優れた開発環境を提供するものですから、日本にもぜひ紹介したい、根付かせたいと考えました。

海外のソフトを日本のマーケットに提供する場合に、コンセプトもなくワープロからゲームまで何でも、という考えは持っていないんです。やはり、日本のコンピュータ環境にとって意味の高いもの、新しい価値を付け加えられるようなものを供給していくことが大切なんだと思います。とくに開発関係ではアメリカに優れたものがたくさんあるんですが、それらが日本ではあまり知られていない、使われていないという現状がありますね。そこをもっと活性化できれば、と思います。

それと、アメリカから持って来たものをそのまま売るだけではつまらない。やるからには、単なるディストリビューターの立場にとどまらず、当社独自の、日本のマーケットに向けた技術の付加ですね、それをきちんとやりたいと考えてきました。

具体的には、向こうのソフトは98マシン上では動かないわけですが、98マシンへの移植をメーカーにやらせるのではなく当社でやる。マニュアルも当社で翻訳して出す。またテクニカルサポートも右から左へメーカーに流すのではなく、できるだけ当社で対処する、ということです。当社独自の付加価値を付けて出すということで、当然メーカーから出しているものとは多少違うところが生まれてきます。そこに私たちのやりがいもあるわけですし、また当社の技術力を上げていく力にもなっていくんだろうと考えています。」

日米の文化の違いを乗り越えるのにひと苦労

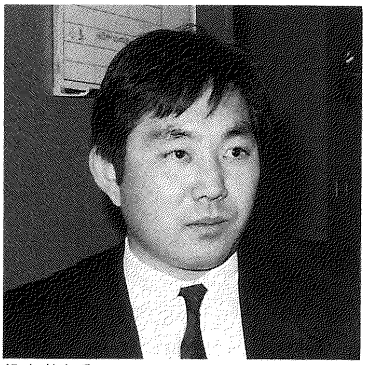
さて、海外のソフトを日本語化する際には、日本のパソコン環境の特殊性など、さまざまな問題がつきまとう。

「海外のメーカーと仕事をしていく中で感じるのは、まず文化の違いですね。日本の商業常識とアメリカの商業常識が違いますから、日本では当然やるようなこともアメリカではやらないといったことがあります。またアメリカではやることも日本ではやらないということもあります。一番苦労しているのは、ライセンス契約でしょうか。日本におけるライセンス契約とぜんぜん違いますので、日本のお客さまに違和感を感じられる部分で、なんとか納得していただくのに苦労することがあります」と松本氏。実際にどのような点が問題となるのか、さらにうかがってみた。

「いままでやってきて驚いたのは、たとえば契約書に〝神に誓いなさい〟とか〝不慮の事故は神のみぞ知る事柄なので、責任はもちません〟とか、そういう条項が出てくることですね。

欧米社会にはキリスト教文化が根深くあるということなんですが、この考え方は日本人にはちょっと理解できないんですね。日本のお客さまにしてみれば、神のみぞ知るじゃ困るよということになります。それをユーザーの方に説明し、納得してもらうのはなかなか難しいことですね。かといって、日本流の契約内容をアメリカのメーカーに要求しても、相手もまた理解を超えていますので、メーカーにとって安心できない内容の契約書になってしまいます。ですから契約書に関しては修正はできないということで、対処しています。

確かにアメリカのビジネスはかなりシビアなんで、ユーザーサポートをきちんとやるという前提は日本もアメリカも同じなんですが、日本ほど〝お客さまは神様だ〟という考え方はないですね。もちろんお客さま第一ではありますが、メーカーとユーザーとの関係のあり方が日本とは違うんです」



松本孝之氏

『386/TOOLBOX』の日本語化で米国メーカーから厚い信頼

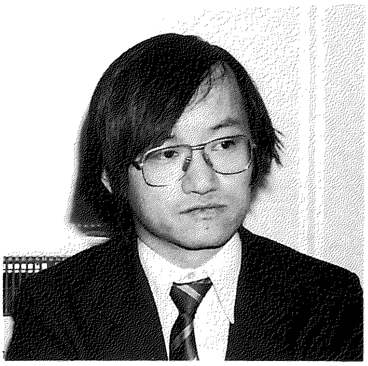
一方、日本語化の作業には98マシンへの移植など、技術面での苦労も多い。菅原裕之さんは入社1年目だが、その技術力を買われてアメリカ側との技術的な調整や最終チェックなどを担当している。

「まずアメリカから持ってきたソフトを移植する際の問題点を調べて、これを調整します。その後移植の作業を経て、最終的にまた私のところで、マニュアル等も含めたチェックをするというのが担当です。

98マシンを中心とした日本独特のパソコン環境は、アメリカでは基本的に理解されていません。そこで日本語処理をする上での日本のパソコンの歴史からひもといて、グラフィックの使い方が違うですとか、そういう説明からしなければならないことも多いですね。

その点をまず理解してもらって、では移植の作業をやりましょうということになるわけですが、たとえば技術資料としてなにがあればできるのかということを調べ上げてアメリカ側に提供したり、また、こちらで作った方が速ければ私が作るということもやっています。開発とマーケティングと連絡員、すべて含めた形で自分で仕事をまとめてゆくところは、苦労はありますが、やりがいがあると感じています」

こうした日米の共同作業を経て日本の市場へ提供されたソフトのひとつに『386/TOOLBOX』があるが、これは32ビットプロテクトモードを管理する386/DOS-Extenderのほか、アセンブラ、リンカ、ライブラリアン等を集大成した総合ユーティリティとして、開発関係で高い評価を受けている。それゆえ同社は、開発元である米国Phar Lap社の厚い信頼をも得ている。



菅原裕之氏

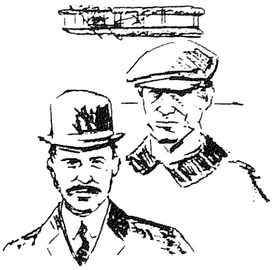
日本における開発環境をもっと豊かで、快適に

最後に、同社システム営業部の今後の展開について、松本氏にうかがった。

「具体的には、オブジェクト言語ですとか、もちろんWindowsやOS/2ですとか、またAIといった分野でも、いいものを提供していきたいと思います。技術的にはあまり後ろは見ないで、一歩先を見ていきたいですね。日本の技術が追いついてアメリカより先に行くような時代がくれば別でしょうが、現在においてはまだまだアメリカのものを紹介し、日本語化して供給していく仕事が必要だと思います。その中で、私どもの技術力もつけて、今度はそのノウハウを生かした日本独自の新しい技術を開発できるように、それをめざしていきたいという気持ちでいます」

日本での開発環境をもっと幅広く、多種多様に提供し、ソフトウエアハウスのみなんさにいいアプリケーションをたくさん作ってもらえれば、と語る両氏。日本におけるアプリケーション文化を高めていく上で、その仕事は重要な役割を担っていると言えるのではないだろうか。

プログラマを求めています。



1903年12月17日、
ノースカロライナのキティホークの丘。
ライト兄弟のフライヤー号は、ふたりの熱い想いと、
人類の夢をのせて大空を舞った。
人に優しい、質の良いソフトウェアを提供し、
コンピューターをもっと人に近づけること。
このソフトヴィジョンの大きな夢は、時代の抱く
夢でもあると思う。

求む！品質管理者
ユーザサポートスタッフ

ソフトヴィジョンは，1989年1月にプログラマが集まって始めた小さなソフトウェアハウスです。「プロの技術に裏付けされた，真に利用者の立場に立ったソフトウェアを」という理想のもと，7人で会社を始めました。

この1年間，プログラマを中心に仲間を募集してきました。プログラマの数が増えてソフトウェアの開発力がついてくるにつれ，ソフトウェアのテストや完成後のユーザサポートに専任スタッフが必要となってきました。今回は，私たちが求めているスタッフの業務内容を少し具体的に書いてみます。もちろん，これまでどおりプログラマも求めています。

アプリケーションソフトウェアの品質管理

全体としてはソフトウェア開発チームの中にあって，しかし，プログラマや内部設計者とは別の「ユーザとして」の観点で，仕様の細部を決め，完成したソフトウェアがその仕様通りに動くかテストし，実際に使ってみて仕様が適切であったかどうか判断し，必要な修正をプログラマに指示することによって，最終的にソフトウェアを完成させる。これが品質管理の仕事です。もちろん，完成後のソフトウェアに不具合や改善点が見つかれば，修正を検討し，指示することも仕事のうちです。品質管理の仕事をするためには，製作チームと密接なコミュニケーションをとるためにソフトウェア開発に関する専門的な知識が必要であり，かといって製作チームに埋没して馴れ合いにならないよう，立場としてはユーザの観点に立ってソフトウェアを見ることが重要です。

対象ソフトウェアは，MS-Windows上のアプリケーションソフトウェアが主体です。こんな仕事をやってみたい方，是非仲間に加わってください。

ユーザサポート

私たちの会社はプログラマばかりが集まって作ってしまったために，ソフトウェアを作ることは得意なのですが，作ったソフトウェアをより良く使っていただくということに関して不十分であったと反省しています。また，今後は多数のユーザを対象にしたパッケージソフトウェアの開発・販売を拡大していく予定です。これらに対処するために，品質管理よりもさらに一歩ユーザ側に寄ったサポートスタッフにも専任者を募集します。

ユーザのソフトウェア使用環境は極めて多種多様です。それらすべてに対する説明をマニュアルに記述できるものではありません。また，十分に記述したつもりでも理解していただけないことは多々あります。こんなときにユーザの問合せに答え，また，そのようなことが起きないように各種パブリシティ，セミナーなどを通じてさまざまな情報を提供する，それがサポートの仕事です。

ソフトウェアを作るよりも，いろいろなソフトウェアをとことん使い込んでみるのが好きという人をよく見かけます。そんな方で，しかもお客様の立場に立って問題に対処できる方，是非我が社のユーザサポートを手伝ってください。今回募集する方は，今後徐々に大きくなっていくであろうサポートチームの中心になっていただける方です。よろしくお願いします。

プログラマ

相変らずプログラマが足りません。最近取り組んでいるプログラムは，

Windowsアプリケーション
RTFJを用いた文書ファイル変換プログラム
通信ソフト「マイリンク」の移植，改良

などです。書いてみませんか？

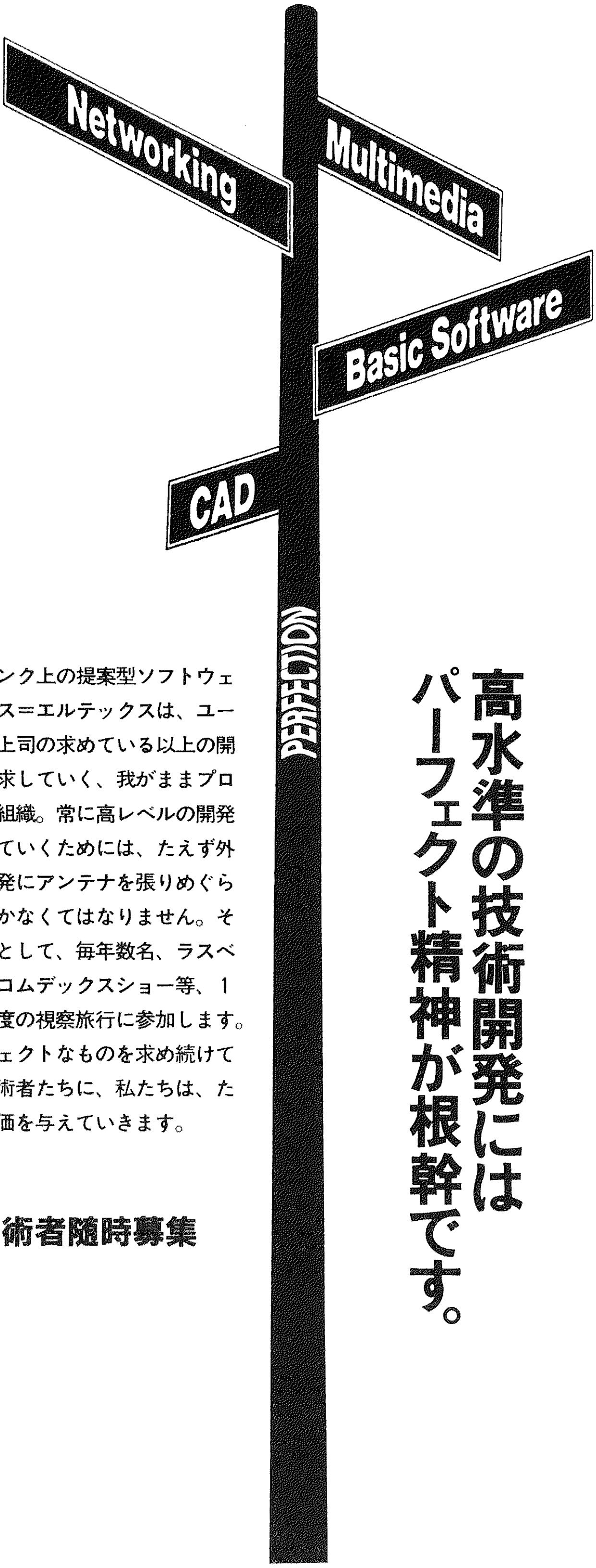
Soft Vision

株式会社ソフトヴィジョン

〒111 東京都台東区浅草橋3-24-8 伊藤ビル
TEL03(5687)8571 FAX03(5687)8718

■**募集要項**●職種／プログラマ(Windows/MS-DOS/UNIX)，プロジェクトマネージャ，技術営業，テクニカルライター，品質管理●資格／大卒以上または実務経験2年以上20～35歳くらいまでの方●給与／経験・能力・職務によります。具体的にはお問い合わせください。●休日休暇／完全週休2日制，祝日，年末年始 有給休暇(初年度15日，連続休暇有)●勤務時間／9:00～17:30フレックスタイム有●応募／お気軽にお問い合せのうえ，履歴書，職務経歴書を郵送またはご持参ください。応募の秘密は厳守いたします。(担当 山中)

■**会社概要**●設立／昭和64年1月5日●資本金／1750万円●従業員／12名●年商／1億5000万円●代表者／取締役社長 酒井俊夫●事業内容／(1)コンピュータソフトウェアの企画・設計・制作 (2)コンピュータ関連の調査・研究 (3)コンピュータ関連のコンサルテーション



高水準の技術開発には
パーフェクト精神が根幹です。

ワンランク上の提案型ソフトウェアハウス＝エルテックスは、ユーザーや上司の求めている以上の開発を追求していく、我がままプロ集団の組織。常に高レベルの開発を行っていくためには、たえず外部の開発にアンテナを張りめぐらしておかなくてはなりません。その一環として、毎年数名、ラスベガスのコムデックスショー等、1週間程度の視察旅行に参加します。パーフェクトなものを求め続けていく技術者たちに、私たちは、たえず評価を与えていきます。

技術者随時募集

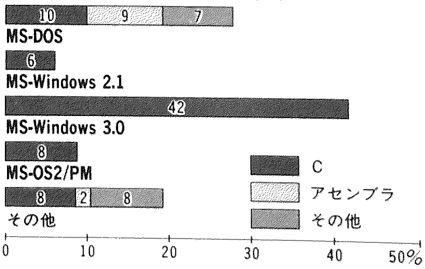
会社概要 ELTEX

設　立	昭和60年12月14日
資本金	1,000万円
事業内容	パソコンを中心としたシステムの提案・開発・販売　①CD-ROM・LD など新しいメディアに対応したシステムの開発 ②ネットワークの端末システムの開発 ③特定業種向けCADの開発・販売 ④基本ソフトの開発
特　徴	大手クライアント向けのシステム開発が多く、ネットワーク関連システムやマルチメディアなど幅広く対応している。また、ＣＡＤを中心とした自社プロダクツにも積極的で、その比率も増加している。基本ソフトの分野では、ハードウェア専門のベンチャー企業とジョイントし、ＯＥＭ製品のＢＩＯＳ・ユーティリティなどを企画から手掛けている。
取引先	三菱電機、三菱自動車、日本航空　他
関連会社	株式会社ソーテック、学校法人福井情報ビジネス総合学院

募集要項 ELTEX

職　種	①システムエンジニア　②プログラマー
資　格	専門卒以上　20～36歳の方　※Ｃ言語・アセンブラ言語経験者大歓迎　※未経験者でも好奇心・向学心の旺盛な方は歓迎します。
給　与	17万円（専門卒初任給）～46万円（36歳課長、諸手当含む） 24歳例：23万5000円（情報処理１種手当・住宅手当含む） 30歳例：31万6000円（中級SE、情報処理１種・住宅手当含む）
手　当	時間外手当、役職手当（5000円～６万円）、住宅手当（１万5000円）、資格手当（２種：5000～特種：２万円）、通勤手当
賞　与	賞与年２回（平成１年度実績年間５ヵ月）、決算賞与年１回（平成１年度実績１ヵ月）
勤務地	本社（横浜駅近く） ◇派遣・出向はありません。
勤務時間	フレックスタイム制（コアタイム10:00～16:00）試用期間中は標準時間9:00～17:45
休日休暇	完全週休２日制（土・日）、祝日、年３回連休（GW・夏期・年末年始）、有給・特別・休暇
福利厚生	各種社会保険完備、退職金制度、財形貯蓄、社員旅行、海外研修、定期健康診断、社員寮、スポーツクラブ法人会員
応募方法	電話連絡の上、履歴書（写真貼付）、職務経歴書を持参し来社ください。郵送も受け付けます。
連絡先	〒220 横浜市西区南幸二丁目20番５号　東伸24ビル５階　℡(045)322-9667　総務課

最近のOS・言語別開発動向



採用HOT・LINE
TEL.(045)322-9667

株式会社エルテックス

MS-DOS Version 5.0ファンクションリクエスト一覧(PC-9801)

ZFはゼロフラグ, CFはキャリーフラグ, それぞれ1でセット, 0でリセットを示す.

00H	プログラムの終了 (呼) AH=00H CS=PSPのセグメントアドレス (返) なし
01H	文字入力(エコーあり) (呼) AH=01H (返) AL=入力された文字
02H	文字出力 (呼) AH=02H DL=出力する文字コード (返) なし
03H	補助入力 (呼) AH=03H (返) AL=補助装置から入力された文字
04H	補助出力 (呼) AH=04H DL=補助装置に出力する文字コード (返) なし
05H	文字のプリンタ出力 (呼) AH=05H DL=プリンタに出力する文字コード (返) なし
06H	直接コンソール入出力 (呼) AH=06H DL=出力する文字コード(DL≠FFHの場合)/FF(入力の場合) (返) なし(コール時DL≠FFHの場合) ZF=0 AL=00H(コール時DL=FFH) ZF=1 AL=キャラクタ(コール時DL=FFH)
07H	直接コンソール文字入力 (呼) AH=07H (返) AL=キーボードから入力された文字
08H	文字入力(エコーなし) (呼) AH=08H (返) AL=キーボードから入力された文字
09H	文字列の表示 (呼) AH=09H DS:DX=出力する文字列のポインタ, 文字列の終わりは'$' (返) なし
0AH	バッファドキーボード入力 (呼) AH=0AH DS:DX=入力バッファのポインタ (返) なし
0BH	キーボードのステータスの検査 (呼) AH=0BH (返) AL=FFH(タイプアヘッドバッファ内に文字が入っている)/00H(タイプアヘッドバッファ内に文字が入っていない)
0CH	バッファを空にして, キーボード入力 (呼) AH=0CH AL=01H,06H,07H,08H,0AH(対応するMS-DOSファンクションリクエストの実行)/ほかの値(これ以上の処理は行わない) (返) AL=00H(タイプアヘッドバッファを空にした)/コール時ALにファンクションリクエストを指定した場合は, そのファンクションの返り値に準ずる
0DH	ディスクのリセット (呼) AH=0DH (返) なし
0EH	ディスクの選択 (呼) AH=0EH DL=ドライブ番号(00H=A:,01H=B:,…) (返) AL=論理ドライブの数
0FH	ファイルのオープン (呼) AH=0FH DS:DX=FCBのポインタ (返) AL=00H(ディレクトリエントリが存在する)/FFH(ディレクトリエントリが存在しない)
10H	ファイルのクローズ (呼) AH=10H DS:DX=FCBのポインタ (返) AL=00H(ディレクトリエントリが存在する)/FFH(ディレクトリエントリが存在しない)
11H	最初のエントリを検索 (呼) AH=11H DS:DX=FCBのポインタ (返) AL=00H(ディレクトリエントリが存在する)/FFH(ディレクトリエントリが存在しない)
12H	次のエントリを検索 (呼) AH=12H DS:DX=FCBのポインタ (返) AL=00H(ディレクトリエントリが存在する)/FFH(ディレクトリエントリが存在しない)
13H	ファイルの削除 (呼) AH=13H DS:DX=FCBのポインタ (返) AL=00H(ディレクトリエントリが存在する)/FFH(ディレクトリエントリが存在しない)
14H	シーケンシャルな読み出し (呼) AH=14H DS:DX=FCBのポインタ (返) AL=00H(正常な読み出し)/01H(EOF)/02H(DTAで示されるバッファが小さすぎる)/03H(EOF, レコードの一部分)
15H	シーケンシャルな書き込み (呼) AH=15H DS:DX=FCBのポインタ (返) AL=00H(正常な書き込み)/01H(ディスクに空き領域がない)/02H(DTAで示されるバッファが小さすぎる)
16H	ファイルの作成 (呼) AH=16H DS:DX=FCBのポインタ (返) AL=00H(空のディレクトリエントリが存在する)/FFH(空のディレクトリエントリが存在しない)
17H	ファイル名の変更 (呼) AH=17H DS:DX=FCBのポインタ (返) AL=00H(ディレクトリエントリが存在する)/FFH(目的のディレクトリエントリが存在しないか, ファイル名がすでに存在する)
19H	カレントドライブ番号の取得 (呼) AH=19H (返) AL=カレントドライブ番号(00H=A:,01H=B:,…)
1AH	DTAの設定 (呼) AH=1AH DS:DX=DTAのポインタ (返) なし
1BH	デフォルトドライブのデータの取得 (呼) AH=1BH (返) AL=1クラスタあたりのセクタ数 CX=1セクタあたりのバイト数 DX=1ドライブあたりのクラスタ数 DS:BX=FAT-IDのポインタ
1CH	ドライブデータの取得 (呼) AH=1CH DL=ドライブ番号(00H=カレント,01H=A:,02H=B:,…) (返) AL=FFH(ドライブ番号の指定が無効)/1クラスタあたりのセクタ数(AL≠FFH) CX=1セクタあたりのバイト数 DX=1ドライブあたりのクラスタ数 DS:BX=FAT-IDのポインタ
21H	ランダムな読み出し (呼) AH=21H DS:DX=FCBのポインタ (返) AL=00H(正常な読み出し)/01H(EOF)/02H(DTAで示されるバッファが小さすぎる)/03H(EOF, レコードの一部分)
22H	ランダムな書き込み (呼) AH=22H DS:DX=FCBのポインタ (返) AL=00H(正常な書き込み)/01H(ディスクに空き領域がない)/02H(DTAで示されるバッファが小さすぎる)
23H	ファイルの大きさの取得 (呼) AH=23H DS:DX=FCBのポインタ (返) AL=00H(ディレクトリエントリが存在する)/FFH(ディレクトリエントリが存在しない)
24H	相対レコードの設定 (呼) AH=24H DS:DX=FCBのポインタ (返) なし
25H	割り込みベクタの設定 (呼) AH=25H AL=割り込みタイプ番号 DS:DX=割り込み処理ルーチンの先頭アドレス (返) なし
26H	新しいPSPの作成 (呼) AH=26H DX=PSPのセグメントアドレス (返) なし
27H	ランダムなブロックの読み出し (呼) AH=27H DS:DX=FCBのポインタ CX=読み出すレコード数 (返) AL=00H(正常な読み出し)/01H(EOF, または空レコード)/02H(DTAで示されるバッファが小さすぎる)/03H(EOF, レコードの一部分) CX=読み取ったレコード数
28H	ランダムなブロックの書き込み (呼) AH=28H DS:DX=FCBのポインタ CX=00H(ファイルサイズフィールドの設定)/書き込むレコード数(CX≠00H) (返) AL=00H(正常な書き込み)/01H(ディスクに空き領域がない)/02H(DTAで示されるバッファが小さすぎる) CX=書き込まれたレコード数
29H	ファイル名の解析 (呼) AH=29H AL=解析の制御 DS:SI=解析する文字列 ES:DI=FCBのポインタ (返) AL=00H(ワイルドカードが使用されていない)/01H(ワイルドカードが使用されている)/FFH(ドライブ文字が無効) DS:SI=解析した文字列の直後のポインタ ES:DI=FCBのポインタ
2AH	日付の取得 (呼) AH=2AH (返) CX=年(1980～2079) DH=月(1～12) DL=日(1～31) AL=曜日(00H=日,01H=月,…06H=土)
2BH	日付の設定 (呼) AH=2BH CX=年(1980～2079) DH=月(1～12) DL=日(1～31) (返) AL=00H(有効な日付)/FFH(無効な日付)
2CH	時刻の取得 (呼) AH=2CH (返) CH=時(0～23) CL=分(0～59) DH=秒(0～59)
2DH	時刻の設定 (呼) AH=2DH CH=時(0～23) CL=分(0～59) DH=秒(0～59) DL=00H (返) AL=00H(有効な時刻)/FFH(無効な時刻)
2EH	ベリファイフラグの設定 (呼) AH=2EH AL=00H(ベリファイを行わない)/01H(ベリファイを行う) DL=00H (返) なし
2FH	DTAの取得 (呼) AH=2FH (返) ES:BX=DTAのポインタ
30H	MS-DOSバージョン番号の取得 (呼) AX=3000H (返) AL=バージョン番号の整数部 AH=バージョン番号の小数部 BH=FFH BL:CX=000000H
31H	プロセスの常駐終了 (呼) AH=31H AL=抜け出しコード DX=パラグラフ(16バイト単位)でのメモリサイズ (返) なし
3300H	CTRL-C検査の設定／解除 (呼) AX=3300H(状態の取得)/3301H(設定) DL=00H(AL=01Hの場合, CTRL-C検査の解除)/01H(AL=01Hの場合, CTRL-C検査の設定) (返) DL=00H(コールしたときAL=00Hの場合, CTRL-C検査が設定されていない)/01H(コールしたときAL=00Hの場合, CTRL-C検査が設定されている) AL=FFH(コールしたときALが00Hまたは01Hでない)
3305H	ブートドライブ番号の取得 (呼) AX=3305H (返) DL=ドライブ番号(01H=A:,02H=B:,…)
3306H	DOSの配置情報の取得 (呼) AX=3306H (返) BL=バージョン番号の整数部 BH=バージョン番号の小数部 DH=MS-DOSのバージョンフラグ/10H(MS-DOSはハイメモリにある) DL=下位の3ビットにリビジョン番号, ほかのビットは0
35H	割り込みベクタの取得 (呼) AH=35H AL=割り込み番号 (返) ES:BX=割り込みルーチンのポインタ
36H	ディスクのフリースペースの取得 (呼) AH=36H DL=ドライブ番号(00H=カレント,01H=A:,02H=B:,…) (返) BX=使用可能なクラスタ数 DX=1ドライブあたりのクラスタ数 CX=1セクタあたりのバイト数 AX=1クラスタあたりのセクタ数(AX≠FFFFH)/FFFFH(ドライブ番号が無効)
38H	国別情報の取得 (呼) AH=38H AL=国のカントリーコード(00H=現在の国, 01H=USA, 51H=日本) DS:DX=情報に対する32バイトのバッファのポインタ (返) CF=0(DS:DXで指定したバッファに情報が設定された) CF=1 AX=01H(カントリーコードが無効) 国別情報の設定 (呼) AH=38H DX=FFFFH AL=FFH(BXにFFHより大きなカントリーコードが設定されている場合)/カントリーコード(AL≠FFH) BX=カントリコード(AL=FFHの場合) (返) CF=0(エラーなし) CF=1 AX=01H
39H	ディレクトリの作成 (呼) AH=39H DS:DX=パス名のポインタ (返) CF=0(エラーなし) CF=1 AX=03H/05H
3AH	ディレクトリの削除 (呼) AH=3AH DS:DX=パス名のポインタ (返) CF=0(エラーなし) CF=1 AX=03H/05H/10H
3BH	カレントディレクトリの変更 (呼) AH=3BH DS:DX=パス名のポインタ (返) CF=0(エラーなし) CF=1 AX=03H
3CH	ハンドルを使うファイル名の作成 (呼) AH=3CH DS:DX=パス名のポインタ CX=ファイルの属性 (返) CF=0 AX=ファイルハンドル CF=1 AX=03H/04H/05H
3DH	ハンドルを使うファイルのオープン (呼) AH=3DH AL=ファイルアクセスコントロール DS:DX=パス名のポインタ (返) CF=0 AX=ファイルハンドル CF=1 AX=01H/02H/03H/04H/05H/0CH

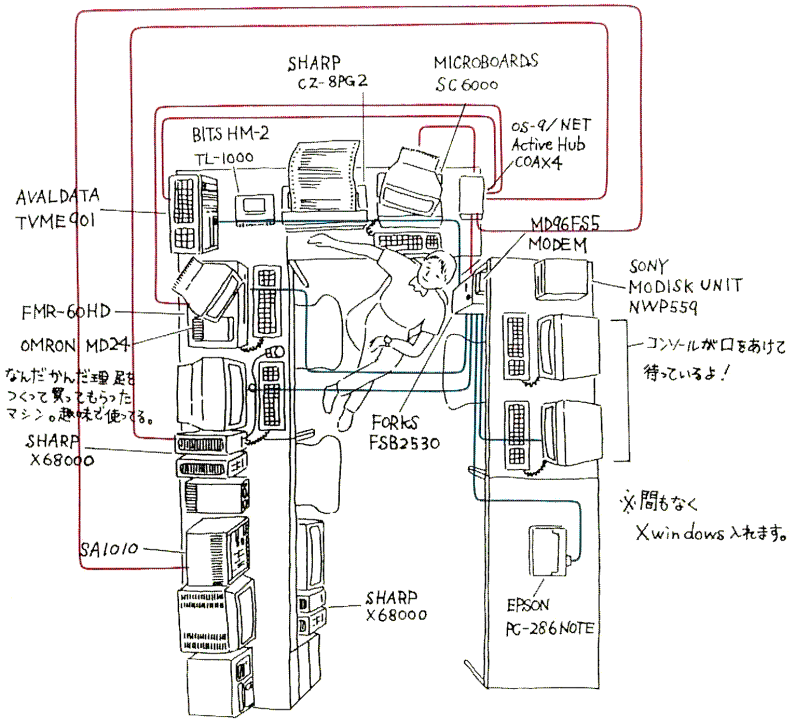
3EH	ハンドルを使うファイルのクローズ (呼) AH=3EH BX=ファイルハンドル (返) CF=0(エラーなし) CF=1 AX=06H
3FH	ファイルかデバイスの読み出し (呼) AH=3FH DS:DX=バッファのポインタ CX=読み出すバイト数 BX=ファイルハンドル (返) CF=0 AX=読み出されたバイト数 CF=1 AX=05H/06H
40H	ファイルかデバイスへの書き込み (呼) AH=40H DS:DX=バッファのポインタ CX=書き込むバイト数 BX=ファイルハンドル (返) CF=0 AX=書き込まれたバイト数 CF=1 AX=05H/06H
41H	ディレクトリエントリの削除 (呼) AH=41H DS:DX=パス名のポインタ (返) CF=0(エラーなし) CF=1 AX=02H/05H
42H	ファイルポインタの移動 (呼) AH=42H CX:DX=オフセット(移動するバイト数) AL=移動方法 BX=ファイルハンドル (返) CF=0 DX:AX=新規のポインタロケーション CF=1 AX=01H/06H
43H	ファイル属性の取得/設定 (呼) AH=43H DS:DX=パス名のポインタ AL=00H(属性を取得する)/01H(属性を設定する) CX=設定する属性(AL=01Hの場合) (返) CF=0 CX=属性(AL=00Hの場合) CF=1 AX=01H/02H/03H/05H
4400H	IOCTLデータの取得 (呼) AX=4400H BX=ハンドル (返) CF=0 DX=デバイスデータ CF=1 AX=01H/06H
4401H	IOCTLデータの設定 (呼) AX=4401H BX=ハンドル DX=デバイスデータ(DH=00H) (返) CF=0 DX=デバイスデータ CF=1 AX=01H/06H
4402H	IOCTLキャラクタを受け取る (呼) AX=4402H BX=ハンドル CX=コントロールデータのバイト数 DS:DX=バッファのポインタ (返) CF=0 AX=転送されたバイト数 CF=1 AX=01H/06H
4403H	IOCTLキャラクタを送る (呼) AX=4403H BX=ハンドル CX=コントロールデータのバイト数 DS:DX=バッファのポインタ (返) CF=0 AX=転送されたバイト数 CF=1 AX=01H/06H
4404H	IOCTLブロックを受け取る (呼) AX=4404H BL=ドライブ番号(00H=カレント,01H=A:, 02H=B:…) CX=コントロールデータのバイト数 DS:DX=バッファのポインタ (返) CF=0 AX=転送されたバイト数 CF=1 AX=01H/05H
4405H	IOCTLブロックを送る (呼) AX=4405H BL=ドライブ番号(00H=カレント,01H=A:, 02H=B:…) CX=コントロールデータのバイト数 DS:DX=バッファのポインタ (返) CF=0 AX=転送されたバイト数 CF=1 AX=01H/05H
4406H	入力ステータスのチェック (呼) AX=4406H BX=ハンドル (返) CF=0 AL=00H(レディ状態ではない)/FFH(レディ) CF=1 AX=01H/05H/06H/0DH
4407H	出力ステータスのチェック (呼) AX=4407H BX=ハンドル (返) CF=0 AL=00H(レディでない)/FFH(レディ) CF=1 AX=01H/05H/06H/0DH
4408H	IOCTLが交換可能か調べる (呼) AX=4408H BL=ドライブ番号(00H=カレント,01H=A:, 02H=B:…) (返) CF=0 AX=00H(交換可能)/01H(交換不可能) CF=1 AX=01H/0FH
4409H	IOCTLリモートブロックデバイスの検出 (呼) AX=4409H BL=ドライブ番号(00H=カレント,01H=A:, 02H=B:…) (返) CF=0 DX=デバイス属性ワード CF=1 AX=01H/0FH
440AH	IOCTLリモートハンドルの検出 (呼) AX=440AH BX=ハンドル (返) CF=0 DX=IOCTLビットフィールド CF=1 AX=01H/06H
440BH	IOCTLリトライ回数の設定 (呼) AX=440BH DX=リトライの回数 CX=待ち時間 (返) CF=0(エラーなし) CF=1 AX=01H
440CH	一般IOCTL(ハンドル用) (呼) AX=440CH BX=ハンドル CH=05H(カテゴリーコード) CL=ファンクション(マイナー)コード DS:DX=データバッファへのポインタ (返) CF=0(エラーなし) CF=1 AX=01H
440DH	一般IOCTL(ブロックデバイス用) (呼) AX=440DH BL=デバイス番号(00H=カレント,01H=A:, 02H=B:…) CH=08H(カテゴリコード) CL=ファンクション(マイナー)コード DS:DX=パラメータブロック-1へのポインタ (返) CF=0(エラーなし) CF=1 AX=01H/05H/0FH
440E ,0FH	論理ドライブマップの取得/設定 (呼) AH=44H AL=0EH(論理ドライブマップの取得)/0FH(論理ドライブマップの設定) BX=ドライブ番号(00H=カレント,01H=A:, 02H=B:…) (返) CF=0 AL=00H(物理的にマップされている) CF=1 AX=01H/0FH
4410H	プリンタサポートの問い合わせ (呼) AX=4410H BX=ファイルハンドル CH=05H CL=ファンクションコード (45H:繰返カウントの設定,65H:繰返カウントの取得) (返) CF=0(エラーなし) CF=1 AX=01H
4411H	IOCTLデバイスの問い合わせ (呼) AX=4411H BX=ドライブ番号(00H=カレント,01H=A:, 02H=B:…) CH=08H CL=ファンクションコード(40H:デバイスパラメータの設定,41H:論理ドライブ上で書き込み,42H:論理ドライブ上でフォーマット,46H:メディアIDの設定,60H:デバイスパラメータの取得,61H:論理ドライブ上で読み込み,62H:論理ドライブ上でベリファイ,66H:メディアIDの取得) (返) CF=0(エラーなし) CF=1 AX=01H
45H	ファイルハンドルの二重化 (呼) AH=45H BX=ファイルハンドル (返) CF=0 AX=新規のファイルハンドル CF=1 AX=04H/06H
46H	ファイルハンドルの強制二重化 (呼) AH=46H BX=既存のファイルハンドル CX=新規のファイルハンドル (返) CF=0(エラーなし) CF=1 AX=04H/06H
47H	カレントディレクトリの取得 (呼) AH=47H DS:SI=64バイトのメモリ領域に対するポインタ DL=ドライブ番号(00H=カレント,01H=A:, 02H=B:…) (返) CF=0(エラーなし) CF=1 AX=0FH
48H	メモリの割り当て (呼) AH=48H BX=割り当てるメモリサイズ(パラグラフ) (返) CF=0 AX=割り当てられたメモリのセグメントアドレス(パラグラフ) CF=1 AX=07H/08H BX=割り当て可能な最大メモリサイズ
49H	割り当てられたメモリの解放 (呼) AH=49H ES=解放するメモリのセグメントアドレス (返) CF=0(エラーなし) CF=1 AX=07H/09H
4AH	割り当てられたメモリブロックの変更 (呼) AH=4AH ES=変更するメモリのセグメントアドレス BX=変更するメモリサイズ(パラグラフ) (返) CF=0(エラーなし) CF=1 AX=07H/08H/09H BX=割り当て可能なメモリサイズ
4B00H	プログラムのロードと実行 (呼) AX=4B00H DS:DX=パス名のポインタ ES:BX=パラメータブロックのポインタ (返) CF=0(エラーなし) CF=1 AX=01H/02H/04H/05H/08H/0AH/0BH
4B03H	オーバーレイのロード (呼) AX=4B03H DS:DX=パス名のポインタ ES:BX=パラメータブロックのポインタ (返) CF=0(エラーなし) CF=1 AX=01H/02H/04H/05H/0AH
4B05H	新しいプログラムの実行準備 (呼) AX=4B05H DS:DX=EXECSTATE構造体のポインタ (返) なし
4CH	プロセスの終了 (呼) AH=4CH AL=リターンコード (返) なし
4DH	子プロセスからリターンコードを取得 (呼) AH=4DH (返) AX=抜け出しコード
4EH	最初に一致するファイル名の検索 (呼) AH=4EH DS:DX=パス名のポインタ CX=属性 (返) CF=0(エラーなし) CF=1 AX=02H/12H
4FH	つぎに一致するファイル名の検索 (呼) AH=4FH (返) CF=0(エラーなし) CF=1 AX=12H
54H	ベリファイステータスの取得 (呼) AH=54H (返) AL=00H(ベリファイオフ)/01H(ベリファイオン)
56H	ディレクトリエントリの変更 (呼) AH=56H DS:DX=パス名のポインタ ES:DI=新規のパス名のポインタ (返) CF=0(エラーなし) CF=1 AX=02H/05H/11H
57H	ファイルの日付/時刻の取得/設定 (呼) AH=57H AL=00H(日付/時刻の取得)/01H(日付/時刻の設定) BX=ファイルハンドル CX=設定する時刻(AL=01Hの場合) DX=設定する日付(AL=01Hの場合) (返) CF=0 AL=01H(エラーなし) AL=00H CX=時刻/DX=日付 CF=1 AX=01H/06H
5800H	アローケーションストラテジの取得 (呼) AX=5800H (返) CF=0 AX=アロケーションストラテジの値 CF=1 AX=01H
5801H	アローケーションストラテジの設定 (呼) AX=5801H BX=アロケーションストラテジ (返) CF=0(エラーなし) CF=1 AX=01H
5802H	上位メモリブロックのリンク状態の取得 (呼) AX=5802H (返) AL=00H(上位メモリブロックがリンクされていない)/01H(上位メモリブロックがリンクされている)
5803H	上位メモリブロックのリンク状態の設定 (呼) AX=5803H BX=00H(上位メモリブロックをリンクしない)/01H(上位メモリブロックをリンクする) (返) CF=0(エラーなし) CF=1 AX=01H/07H
59H	拡張エラーコードの取得 (呼) AH=59H BX=00H (返) AX=拡張されたエラーコード BH=エラークラス BL=可能な対処 CH=エラーの発生した場所/CL,DX,SI,DI,BP,DS,ESの各レジスタの内容は破壊される
5AH	一時ファイルの作成 (呼) AH=5AH CX=属性 DS:DX=パス名のポインタ (返) CF=0 AX=ファイルハンドル CF=1 AX=03H/05H
5BH	新しいファイルの作成 (呼) AH=5BH CX=属性 DS:DX=パス名のポインタ (返) CF=0 AX=ファイルハンドル CF=1 AX=03H/04H/05H/50H
5C00H	ファイルアクセスロック (呼) AX=5C00H BX=ファイルハンドル CX:DX=ロックされた領域のポインタ SI:DI=ロックされた領域のサイズ (返) CF=0(エラーなし) CF=1 AX=01H/06H/21H
5C01H	ファイルアクセスロック解除 (呼) AX=5C01H BX=ハンドル CX:DX=ロックを解除する領域のポインタ SI:DI=ロックを解除する領域のサイズ (返) CF=0(エラーなし) CF=1 AX=01H/06H/21H
5E00H	マシン名の取得 (呼) AX=5E00H DS:DX=バッファのポインタ (返) CF=0 CX=ローカルコンピュータの番号 CF=1 AX=01H
5E02H	プリンタセットアップ (呼) AX=5E02H BX=割り当てリストのインデックス CX=セットアップ文字列のサイズ DS:SI=セットアップ文字列のポインタ (返) CF=0(エラーなし) CF=1 AX=01H
5F02H	割り当てリストのエントリの取得 (呼) AX=5F02H BX=割り当てリストのインデックス DS:SI=ローカル名のためのバッファのポインタ ES:DI=リモート名のためのバッファのポインタ (返) CF=0 BL=03H(プリンタ)/04H(ドライブ) CX=ユーザー変数域 CF=1 AX=01H/12H
5F03H	割り当てリストのエントリの作成 (呼) AX=5F03H BL=03H(プリンタ)/04H(ドライブ) CX=ユーザー変数域 DS:SI=ソースデバイス名のポインタ ES:DI=ディスティネーションデバイス名のポインタ (返) CF=0(エラーなし) CF=1 AX=01H/03H/05H/08H
5F04H	割り当てリストのエントリの取り消し (呼) AX=5F04H DS:SI=ソースデバイス名のポインタ (返) CF=0(エラーなし) CF=1 AX=01H/0FH
62H	PSPアドレスの取得 (呼) AH=62H (返) BX=カレントプロセスのPSPのセグメントアドレス
67H	オープン可能な最大ハンドル数の設定 (呼) AH=67H BX=設定する最大ハンドル数 (返) なし
68H	ファイルのコミット (呼) AH=68H BX=フラッシュするファイルのハンドル (返) CF=0(エラーなし) CF=1 AX=02H
6CH	拡張されたファイルのオープン/クリエイト (呼) AH=6CH BX=ファイルをオープンするときのモード CX=ファイルのアトリビュート DX=アクション DS:SI=パス名の位置 (返) CF=0(エラーなし) CF=1 AX=03H/04H/05H

(返)がCF=1の場合AXに返るエラーコードは以下の通り

01H(ファンクションコードが無効)/02H(ファイルが見つからない)/03H(パス名が見つからない)/04H(ファイルをオープンしすぎている)/05H(アクセスできない)/06H(ハンドルが無効)/07H(メモリコントロールブロックが破損)/08H(メモリが足りない)/09H(メモリブロックアドレスが無効)/0AH(環境が無効)/0BH(書式が無効)/0CH(アクセスコードが無効)/0DH(データが無効)/0EH(予約)/0FH(ドライブ名が無効)/10H(カレントディレクトリを削除しようとした)/11H(同じデバイスでない)/12H(これ以上ファイルがない)/13H(ディスクがライトプロテクトされている)/14H(ディスクユニットが不良)/15H(ドライブが準備されていない)/16H(ディスクコマンドが無効)/17H(CRCエラー)/18H(長さが無効)/19H(シークエラー)/1AH(MS-DOSのディスクでない)/1BH(セクタが見つからない)/1CH(紙切れ)/1DH(書き込み失敗)/1EH(読みだし失敗)/1FH(一般的な失敗)/20H(共有違反)/21H(ロック違反)/22H(ディスクが不正)/23H(FCB使用不可能)/24H〜31H(予約)/32H(ネットワークリクエストがサポートされていない)/33H(リモートコンピュータがLISTEN状態でない)/34H(ネットワーク名が重複している)/35H(ネットワーク名が見つからない)/36H(ネットワークビジー)/37H(ネットワークのデバイスはこれ以上ない)/38H(ネットワークBIOSの限界を超えた)/39H(ネットワークアダプタのハードエラー)/3AH(ネットワークから不正な応答があった)/3BH(予期できないネットワークエラー)/3CH(リモートアダプタのハードエラー)/3DH(プリント待ち行列がいっぱい)/3EH(待ち行列がいっぱいでない)3FH(プリントファイルのためのスペースが足りない)/40H(ネットワーク名は既に削除されている)/41H(アクセスできない)42H(ネットワークデバイスのタイプが不正)43H(ネットワーク名が見つからない)/44H(ネットワーク名の限界を超えた)45H(ネットワークBIOSセッションの限界を超えた)/46H(一時休止)47H(ネットワークの要求が受け付けられない)48H(プリンタかディスクのリダイレクトの中止)/49H〜4FH(予約)/50H(ファイルが存在する)/51H(予約)/52H(作成不能)/53H(割り込みタイプ24Hの失敗)/54H(ネットワーク構造が不正)/55H(割り当て済み)/56H(パスワードが無効)/57H(パラメータが無効)/58H(ネットワークシステムへの書き込み失敗)/5AH(システム関連ファイルがロードされていない)

SOFT VISION

拘り(こだわ)の美学



上記のイラストの主が関勝成さん(31歳)。　そしてずらりと並んだマシンの山は、彼の良き相棒たち。「社内に於ける時空間、および共有物資の私物化、これこそが自由創造の基本である」と関さん。　先行投資の重要性を、身をもって訴え証明してきた努力の人である。「仕事は遊び」と言いきるその姿勢には、遊びも仕事も徹底的に、それも人の指図を受けずにやらないと気がすまないという、拘りの美学が見える。　電話で社のシステムに侵入、いやログインして深夜自宅からソースファイルをつつくなんて事は朝飯前、ついには電話料金節約のために、ビルの屋上にアンテナを立て「これからはパケット通信の時代だ」と吠えながら、空中にエラーメッセージをふりまくというパフォーマンスまで見せたという。　突然夜中にラーメンを食べるためだけに千駄ヶ谷までクルマをとばし、あぁうまかったと言いながら帰って来て、再びキーボードに向かっていたかと思えば、VMEのマニュアルを枕に、ガーガー寝ていたりする。　そんな彼の左手には、文字盤のない腕時計が静かに時を刻んでいる。

会社プロフィール

設立●昭和59年4月2日　代表者●代表取締役　鈴木和芳　資本金●990万円　従業員数●144名(男:105名、女:39名)　平均年齢●25.5歳　売上高●8億8000万円(平成3年度実績)　事業内容●①FA＆ロボットなどの自動制御　②データ通信＆自動車電話などの通信制御　③TV局向け＆医療向けの映像制御・画像処理　④オンライン端末＆無線端末などのマイクロソフト制御　上記4分野を中心としたソフトウェア・ハードウェアの企画、開発、販売、コンサルティング　主要機種●HP64000、HP9000LAN、PC9801、FMR、Macintosh他多数　主要言語●C、アセンブラ、他　主要取引先●日本電気㈱、日本電気無線電子㈱、日本無線㈱、富士通機電㈱、松下システムエンジニアリング㈱、松下通信工業㈱、京セラ㈱、JRCエンジニアリング㈱、東芝エンジニアリング㈱、昭和電子㈱

募集要項

職種●コンピュータエンジニア　※今春・来春新卒者のお問い合せ歓迎　※学生アルバイト同時募集　資格●学歴不問　26歳位迄　給与●月給　固定給制19万円以上　※経験・年齢・前職を考慮の上、相談により決定します。26歳支給例／31万6500円　アルバイト／時給1000円以上(時間相談に応じます。)　待遇●昇給年1回(3年度実績12%)、賞与年2回(2年度実績7ヵ月)、通勤交通費(2万4000円迄支給)、各種社会保険完備、住宅・家族・精勤・時間外・役職手当、退職金制度、財形貯蓄制度、社内旅行(グアム)、クラブ活動(野球部、テニス部、バスケット部、ウインドサーフィン部　他多数クラブ　※会社から援助金支給)、ラップトップ・レンタル制度(長期間無料)　勤務地●本社(相模原)　勤務時間●フレックスタイム制(コアタイム10:30〜15:30)　※タイムカードはありません。1日標準労働時間8時間　休日休暇●完全週休2日制(土・日)、祝日、夏期・年末年始・GW・有給(初年度8日から)・慶弔休暇　※年間125日＋有給　応募●電話連絡の上、履歴書(写真貼付)・職務経歴書(職歴のない方は自己PR書)を持参ください。郵送も可。　※FAX応募も歓迎します。　※応募の秘密厳守します。　※入社日は相談に応じます。　※新卒者の資料請求歓迎します。　※大学生の会社訪問歓迎します。　交通●JR横浜線相模原駅徒歩5分

採用関係連絡先

〒229　神奈川県相模原市相模原4-3-14相模原第一生命ビル「株式会社ソフトビジョン　人事部人事課」　フリーダイヤル0120-211744　採用担当／海老沢、斉藤
FAX　0427-51-3333

株式会社 ソフトビジョン

BORLAND

The Leader in Object-Oriented Programming

新生ボーランド キックオフ・キャンペーン

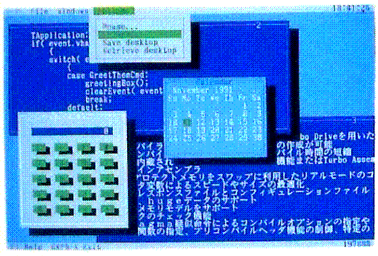
BORLAND KICK OFF CAMPAIGN

いま、店頭でBORLAND C++ 2.0を買うと──

ボーランド純正画面制御ライブラリ アプリケーションフレームワークをプレゼント!!

ただ今、店頭にてBORLAND C++ 2.0をお買い求めになり、付属の「アプリケーションフレームワーク無料送付シール」を登録書に貼って返送していただきますと、アメリカでは$199.95(¥28,000)にて提供されております、ボーランド純正画面制御ライブラリ・アプリケーションフレームワーク(ソースコード付属※注)の98版を無料にて送付させていただきます。高速なコンパイルスピードと大規模なアプリケーション開発に適したプロテクトモードのコンパイラを備え、DOSとWindowsのアプリケーションをCとC++で開発できる、プロの開発者のためのBorland C++。この強力な開発環境に、オブジェクト指向テクノロジーを用い、簡単に画面制御を行えるクラスライブラリ・アプリケーションフレームワークを組み合わせれば、開発環境は飛躍的に向上し、プログラマの作業をより楽にすることができます。

APPLICATION FRAMEWORKS



▲*Turbo Visionを用いたサンプル画面*

プログラマは、自身の創造性やプログラミング能力をアプリケーションの機能や性能の向上のためだけに集中することができるようになります。もう、細かなユーザーインターフェース等のプログラミングに多くの時間をさく必要はありません。Windows 3.0用のObject Windows、DOS用のTurbo Vision。必要なプラットフォームに合わせて2種類のフレームワークから選んでいただけます。どちらもマウス対応のオーバーラップマルチウィンドウやメニュー、スクロールバー、ダイアログ、エディタなど、洗練された機能を提供します。アプリケーションフレームワークを使えば、直感的な操作を可能にし、また構造的に優れたユーザフレンドリィなインターフェースを実現できるのです。

■Object Windows:Windows用の画面制御ライブラリ(ソースコード付属※注)

Object Windowsは、グラフィカル・ユーザインターフェースを実現するために、複雑なAPIと格闘しなければならないという面倒な作業を取り除き、Windows 3.0のアプリケーション開発を合理化します。ボーランド独自の、Windowsメッセージを処理するダイナミックディスパッチの機能により、Windowsプログラミングのモデルを簡素化しているのです。Object Windowsにより、MDI(マルチドキュメントインターフェース)やカスタムコントロールへのアクセスなどのWindowsの特徴的な機能を使用したプログラムを簡単に作成できます。新しくWindowsのプログラミングを始められる方でも、古い方法の経験がある方でも、Object Windowsがプログラミングデザインに提供する新次元の自由度の高さや柔軟性を堪能していただけるでしょう。

■Turbo Vision:DOS用の画面制御ライブラリ(ソースコード付属※注)

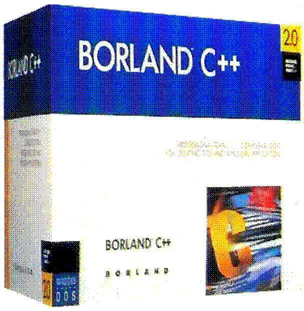
Turbo Visionは、WindowsにおけるObject Windowsと同じように、構造的に優れたユーザインターフェースを、キャラクターベースのDOSアプリケーションに提供します。ほんの少しのコードで、機能的なマウス対応の、位置・サイズ可変なオーバーラップマルチウィンドウやプルダウンメニュー、ダイアログボックス、ボタン等を駆使したイベントドリブン・アプリケーションを、プログラマが望むように実現することができるのです。Turbo Visionを使用すれば、簡単に現代的でエキサイティングなDOSアプリケーションを構築できます。

■価格:BORLAND C++ 2.0アプリケーションフレームワーク・キャンペーンパック…¥98,000(税別)

★価格据え置き!お買い得です!!

※注…ソースコードに関しましては、サポートは一切行われません。ご了承ください。

★さらに、もれなく「プログラミングWINDOWS」(アスキー出版局)をプレゼント!!



BORLAND® C++ 2.0

※All Borland Products are trademarks or registered trademarks of Borland International, Inc. ©Borland International, Inc
※その他、商品名は一般に各社の商標です。
●製品に関するお問い合せは、ボーランド株式会社 TEL.03-5350-9370までお願いします。

ボーランド株式会社
〒151 東京都渋谷区笹塚1-64-8 笹塚サウスビル TEL.03-5350-9370代
※移転いたしました。

T1017777030986 〈資料請求番号P04〉 雑誌コード17777-03